



The author of this thesis has granted the University of Calgary a non-exclusive license to reproduce and distribute copies of this thesis to users of the University of Calgary Archives.

Copyright remains with the author.

Theses and dissertations available in the University of Calgary Institutional Repository are solely for the purpose of private study and research. They may not be copied or reproduced, except as permitted by copyright laws, without written authority of the copyright owner. Any commercial use or publication is strictly prohibited.

The original Partial Copyright License attesting to these terms and signed by the author of this thesis may be found in the original print version of the thesis, held by the University of Calgary Archives.

The thesis approval page signed by the examining committee may also be found in the original print version of the thesis held in the University of Calgary Archives.

Please contact the University of Calgary Archives for further information,

E-mail: uarc@ucalgary.ca

Telephone: (403) 220-7271

Website: <http://www.ucalgary.ca/archives/>

UNIVERSITY OF CALGARY

Design and Construction of an Infra-Red Camera for Use at the
Rothney Astrophysical Observatory

by

Susanna Elaine Johnson

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE
DEGREE OF MASTER OF SCIENCE IN ASTROPHYSICS

DEPARTMENT OF PHYSICS AND ASTRONOMY

CALGARY, ALBERTA

DECEMBER, 2001

© Susanna Elaine Johnson, 2001

ABSTRACT

This thesis describes the design and construction of an infrared camera for astronomical use at the Rothney Astrophysical Observatory, based on a 128 by 128 pixel IR array. The array is sensitive in the 2.0 μm to 5.0 μm range. Construction details for the detector itself are given, and details of the software for the array controller and for the host computer are given, with source code listings.

Acknowledgements:

First, the technical acknowledgements. There is Fred Babott, who is the technical cohesive force behind the RAO and who knows his onions. He did a great deal of the circuit design, layout, and fabrication. Then there is Dr. David Fry, who let me have my head but was able to direct my energies in the appropriate direction. Finally, there is Pat Irwin who allowed me to commandeer laboratory space in which to assemble the IR camera. These three gentlemen are the unsung heroes of the Physics Department at the University of Calgary.

Then, the personal acknowledgements. There are Richard and Patricia Pierson, who funded this effort and provided moral support, and Sybil Lemyre, who listened to me at all hours and under all circumstances. Finally, there is my husband, Larry Harding, who put up with everything that went on and who still manfully refrained from committing justifiable homicide.

Dedication:

To all those who gave comfort, support, love, encouragement, and hope to
an abused and frightened child.

TABLE OF CONTENTS

Approval page	ii
Abstract	iii
Acknowledgements	iv
Dedication	v
Table of Contents	vi
List of Tables	xiii
List of Figures	xiv
Glossary	xvi
Epigraph	xxiii
 CHAPTER ONE: INTRODUCTION	 1
1.0 Introduction	3
 CHAPTER TWO: GENERAL CHARACTERISTICS OF DATA ACQUISITION SYSTEMS	 16
2.0 Introduction	18
2.1 Requirements for an Advanced Data Acquisition System	18
2.2 Elements of an Advanced Data Acquisition System	20
2.2.1 Microprocessor Options	20
2.2.2 Memory Options	23
2.2.2.1 RAM	23
2.2.2.2 ROM	24
2.2.2.3 RMM	25
2.2.2.4 Flash Memory	25
2.2.3 Reconfigurable Logic	26
2.2.4 Building The Platform	28
2.3 IR Arrays and Charge-Coupled Devices	28
2.3.1 Operating Considerations for an IR Array	30
2.4 IR Labs Controller	32
2.4.1 Timing Card	32
2.4.2 Clock Generator Card	33
2.4.3 Analog Input Card	33
2.5 Host	34
2.6 Details of System	34

CHAPTER THREE: IR CAMERA ELECTRONICS AND CABLING	35
3.0 Introduction	37
3.1 Dewar Electronics	37
3.1.1 Rockwell TCM-1000C Array	38
3.1.2 Dewar Electronics	40
3.1.3 Microswitch	40
3.2 RAO Designed Support Electronics	42
3.2.1 Preamplifier	43
3.2.2 Bias Generator Board	44
3.2.3 Logic Board	45
3.2.4 Solenoid Driver Board	45
3.3 IR Labs Controller	46
3.3.1 Timing Board	47
3.3.2. Clock Generator Board	49
3.3.3 Analog Input Board	50
3.3.4 Utility Board	51
3.4 Spectral Instruments Model 1826 PCI CARD	52
3.5 Cabling	52
CHAPTER FOUR: MECHANICAL ASSEMBLY OF DEWAR	54
4.0 Introduction	56
4.1 Filter Wheel Assembly	56
4.2 Cold Plate	60
4.3 IR Array Assembly	61
4.4 Inner Assembly	63
4.5 Outer Assembly	70
4.6 Final Assembly	73

CHAPTER FIVE: GENERAL CHARACTERISTICS OF A REAL- TIME SYSTEM.....	75
5.0 Introduction.....	77
5.1 Environment.....	77
5.2 Definition of Real Time Operation.....	78
5.2.1 Time Constant.....	79
5.2.2 Ability to Stay in Real-Time.....	79
5.3 Mission of the Real-Time System.....	79
5.3.1 Control of the Process.....	79
5.3.2 Control of the Instrument.....	80
5.3.3 Take Data.....	80
5.3.4 Interpret Commands.....	80
5.3.5 Respond to Real-World-Events.....	80
5.3.6 Respond to the Communication Subsystem.....	81
5.3.7 Respond to NMI Emergencies.....	81
5.4 Internal Structure of the Real-Time System.....	81
5.4.1 Interface to Hardware (BIOS).....	82
5.4.1.1 Physical I/O Control Subsystem.....	82
5.4.1.2 Logical I/O Control Subsystem.....	83
5.4.2 Interface to Communication Subsystem.....	83
5.4.2.1 Tick-Tock.....	85
5.4.2.1.1 Transmission.....	85
5.4.2.1.2 Receiving.....	87
5.4.3 Message Cracker.....	86
5.4.3.1 Message Cracker Takes Direct Action.....	87
5.4.3.2 Message Cracker Sets Event Queue.....	88
5.4.4 Interrupt Handlers.....	89
5.4.4.1 Clock/Timer Interrupt Service Routine.....	90
5.4.5 Process Control Routine.....	91
5.4.6 Background Loop.....	92
5.4.7 Virtual Machine.....	93
5.5 Commercially Available Real-Time Operating Systems.....	94
5.6 Roll-Your-Own Operating Systems.....	96

CHAPTER SIX: IR LABS CONTROLLER PROGRAM	97
6.0 Introduction	99
6.1 Structure of DSP Program	102
6.1.1 Loader	103
6.1.2 Kernel	104
6.1.2.1 Communication Subsystem	104
6.1.2.2 Timer Driver	107
6.1.2.3 Load New Overlay Function	108
6.1.2.4 Load Word	108
6.1.2.5 Read Word	108
6.1.3 Application Program	109
6.1.3.1 PON - Power On	111
6.1.3.2 POF - Power Off	112
6.1.3.3 SBV - Set Bias Voltage	112
6.1.3.4 SB2 - Set Video Board Video Offset	112
6.1.3.5 SGN - Set Gain	113
6.1.3.6 STP - Stop Video Mode	113
6.1.3.7 RDA - Read Array	113
6.1.3.8 RRR - Read, Reset, Read	114
6.1.3.9 MRA - Multiple Read of Array	114
6.1.3.10 ABR - Abort Read	115
6.1.3.11 DON - Done	115
6.1.3.12 TST - Communication Test	115
6.1.3.13 SEX - Set Exposure Time	116
6.1.3.14 LDW - Load Word	116
6.1.3.15 OSH - Open Shutter	116
6.1.3.16 CSH - Close Shutter	117

CHAPTER SEVEN: HOST PROGRAM.....	118
7.0 Introduction.....	120
7.1 Program Structure.....	120
7.1.1 Structure of Graphic User Interface.....	122
7.1.2 Structure of the Background Loop.....	123
7.1.3 Structure of Communication Subsystem.....	125
7.1.3.1 Interface to Spectral Instruments PCI-1826 Card.....	125
7.1.3.2 Serial Communication Channel.....	126
7.1.3.3 DMA Channel.....	127
7.1.4 Real-Time Clock.....	127
7.1.5 Image Storage.....	128
7.1.6 Filter Wheel Control.....	129
7.2 Host Program Auxiliary Systems.....	129
7.2.1 Help Subsystem.....	129
7.3 Dynamic Linked Libraries.....	130
7.3.1 Generating LIB files from DLL Files.....	130
7.3.2 FITS Library.....	131
7.4 ACCESS I/O DIO-128.....	132
7.5 White Rat.....	133
7.6 Errors.....	134
7.6.1 Fatal Error.....	134
7.6.2 Correctable Error.....	135
7.6.3 FITS Library Error.....	135
CHAPTER EIGHT: CONCLUSION.....	136
8.0 Overview.....	138
8.1 Status As Of This Writing.....	139
8.2 There And Back Again.....	139
8.2.1 The Controller Incident.....	141
8.2.2 The Computer Factory.....	144
8.2.3 Problems with the IR Labs Controller.....	145
8.2.4 Problems with the Spectral Instruments 1826 Interface Card.....	145
8.2.5 Problems with the IR Labs Host Program.....	146
8.2.6 Problems with the DSP56002 Software.....	146
8.3 Recommended Work.....	147
8.4 Summary.....	148

REFERENCES	150
APPENDIX A: Schematics for RAO Designed Electronics Package	160
Figure A-1. Infrared Detector Shutter and Mounting Plate	161
Figure A-2. Wiring Harness Card for Dewar Electronics	162
Figure A-3. Preamplifier and Bias Voltage Board Layouts	163
Figure A-4. Preamplifier Schematic	164
Figure A-5. Bias Voltage Generator Schematic	165
Figure A-6. Logic Board Layout	166
Figure A-7. Logic Board Schematic (partial)	167
Figure A-8. Shutter Solenoid Driver Board Schematic	168
APPENDIX B: Dewar Connector Pinouts and IR Labs Controller	
Pinout	169
B.1 External Preamplifier Connector	170
B.2 Dewar to Preamplifier Connector	171
B.3 Clock Generator Board DB-37 Connector	172
B.4 DB-37 for Controller to External Preamplifier Pinout	173
B.5 DB-25 for Controller to Filter Wheel Controller Pinout	174
APPENDIX C: Narrow Band Filter Characteristics	175
C.1 iH Filter	176
C.2 iK Filter	177
C.3 iL Filter	178
C.4 Standard M Filter	179
C.5 TCM-1000C Response Curve with iK, iLp, and M Filter	
Passbands Overlaid	180

APPENDIX D: Source Code Listings	182
D.1 Host Program	183
D.1.1 List of Program Files in Host Program	183
D.1.2 Listing of host.Prj	185
D.1.3 Listing of dllmak.prj	196
D.1.4 Listing of host.c	199
D.1.5 Listing of xlate.c	288
D.1.6 Listing of xlate.h	291
D.1.7 Listing of global.h	292
D.1.8 Listing of svid.h	293
D.2 DSP Program	300
D.2.1 Bootstrap.asm as supplied by IR Labs	300
D.2.2 App.asm as supplied by IR Labs	314
D.2.3 Bootstrap.asm as modified by University of Calgary	329
D.2.4 App.asm as modified by University of Calgary	344
D.2.5 Assembly Script for Motorola Assembler Under MS-DOS	363
APPENDIX E: Host Program Error Codes	364
APPENDIX F: FITS Error Codes	369

LIST OF TABLES

Table 2-1. Examples of Microprocessor Architecture.....	22
Table 3-1. Design Criteria for Hybrid Focal Plane Arrays.....	39
Table 6-1. Tasks Available in RAO Application.....	111
Table B-1. External Preamplifier Pinout.....	170
Table B-2. Dewar to Preamplifier Pinout.....	171
Table B-3. Clock Generator Board Pinout.....	172
Table B-4. DB-37 for Controller to External Preamplifier Pinout.....	173
Table B-5. DB-25 for Controller to Filter Wheel Controller Pinout.....	174
Table C-1. Central Wavelengths and Zero Magnitude Fluxes at Different Photometric Bands.....	181

LIST OF FIGURES

Figure 1-1. NGC 4631 in Visible Light.....	5
Figure 1-2. Trapezium in L-Band.....	8
Figure 1-3. Contour Maps of Equivalent Visual Extinction, Northern Streamer.....	10
Figure 1-4. Solar Flare at Near-IR Wavelengths, 6 September 2001.....	12
Figure 1-5. PICNIC Spectral Response Curve.....	14
Figure 1-6. Relative Response as a Function of Wavelength for TCM-1000C IR Array.....	15
Figure 2-1. Block Diagram of Data Acquisition System.....	19
Figure 2-2. Types of Microprocessors.....	21
Figure 3-1. Relative Response as a Function of Wavelength for TCM-1000C IR Array.....	38
Figure 3-2. Schematic of Dewar Electronics from IR Labs Documentation.....	41
Figure 4-1. Schematic Drawing of Filter Wheel.....	57
Figure 4-2. Front View of Filter Wheel Assembly.....	58
Figure 4-3. Reverse Side of Filter Wheel Assembly.....	59
Figure 4-4. Cold Plate.....	61
Figure 4-5. Exploded View of IR Array Assembly.....	62
Figure 4-6. Assembled IR Array Assembly.....	63
Figure 4-7. Filter Wheel Assembly and IR Assembly on Cold Plate.....	64
Figure 4-8. Top View of Dewar with Inner Shell in Place.....	66
Figure 4-9. Outer Shell Bolt Tightening Sequence.....	67
Figure 4-10. Drive Train Mount.....	68
Figure 4-11. Drive Train with Setscrews.....	69
Figure 4-12. Top View of Inner Assembly of IR Camera.....	70
Figure 4-13. Stepper Motor and Reduction Gear Housing.....	71
Figure 4-14. Interior View of Assembled Stepper Motor Reduction Gear Housing.....	72
Figure 4-15. View of Completed Filter Wheel Drive with Stepper Motor, Reduction Gear Housing, and Stepper Motor Controller.....	73
Figure 4-16. Completed Camera in Mount.....	74

Figure 5-1. Block Diagram of Real-Time System.....	78
Figure 5-2. Basic Input/Output Subsystem Block Diagram.....	83
Figure 5-3. Typical Double-Buffered Communication Subsystem.....	84
Figure 5-4. Typical Message Cracker.....	89
Figure 5-5. Typical Interrupt Handler.....	90
Figure 5-6. PID Control Loop Equation.....	93
Figure 5-7. Linkage Sequence for Virtual Machine in RTOS in NRC DRM-210.....	95
Figure 6-1. Motorola 56002 Architecture.....	100
Figure 6-2. Motorola 56000 Family Floating Point/Arithmetic Unit.....	101
Figure 6-3. Loader Block Diagram.....	102
Figure 6-4. Block Structure of Kernel.....	105
Figure 7-1. Host Program Background Loop.....	121
Figure 7-2. Typical Control Window.....	122
Figure 7-3. Program Flow Upon Initialization.....	124
Figure 7-4. Error Screen.....	124
Figure 8-1. Why the Preamplifier is Necessary.....	142
Figure 8-2. Cost Estimate of Proposed In-House Controller Design.....	143
Figure 8-3. Block Diagram, Proposed TMS320C30 Based Controller.....	143
Figure A-1. Infrared Detector Shutter and Mounting Plate.....	160
Figure A-2. Wiring Harness Card for Dewar Electronics.....	161
Figure A-3. Preamplifier and Bias Voltage Board Layouts.....	162
Figure A-4. Preamplifier Schematic.....	163
Figure A-5. Bias Voltage Generator Schematic.....	164
Figure A-6. Logic Board Layout.....	165
Figure A-7. Logic Board Schematic (partial).....	166
Figure A-8. Shutter Solenoid Driver Board Schematic.....	167
Figure C-1. iH Filter Passband.....	173
Figure C-2. iK Filter Passband.....	174
Figure C-3. iL Filter Passband.....	175
Figure C-4. Standard M Band Filter Passband.....	176
Figure C-5. TCM-1000C Response Curve with iK, iLp, and M Filter Passbands Overlaid.....	177

GLOSSARY

2016

An early type of ROM, organized as 8 bits wide by 2 K bytes long. It could be programmed only at the factory.

8051

An INTEL 8-bit microcontroller with a Harvard architecture. It is a very successful design and is widely used. The design specifications may be licenced from INTEL for inclusion as a component in a CPLD or FPGA design.

A/D converter

Analog-to-Digital converter

ADC

See A/D converter

ARCT

A. R. Cross Telescope. The 1.8 metre telescope at the RAO.

ASIC

Application Specific Integrated Circuit. An integrated circuit designed by or for a customer for a specific purpose.

Bang-bang controller

A controller which has two states only: full on and full off.

BCD

Binary Coded Decimal.

Bit Bucket-Brigade

A technique for unloading CCDs and IR arrays in which the charge is moved from pixel to pixel until it arrives at the output port.

Boot-block

A portion of a flash memory that cannot be rewritten under program control. This is usually used to store critical code which if damaged would prevent the system from operating at all.

Canned System

A system purchased from a vendor as a package.

Canned Software

Software purchased from a vendor as a package.

CCD

Charge Coupled Device.

CMOS

Complementary Metal Oxide Semiconductor. A low current-drain logic technology based on FETs instead of junction transistors.

COFF

Common Output File Format. A widely used file format for output from compilers, linkage editors, and other system tools.

COSMAC

A line of RCA microprocessors. Their chief characteristic is their extremely low power consumption, making them ideal for use in handheld or battery powered equipment.

CPLD

Complex Programmable Logic Device.

CRC

Cyclic Redundancy Check. A method of detecting single-bit errors in a data stream.

D/A converter

Digital-to-Analog converter.

DAC

See D/A converter.

DLL
Dynamic Linked Library. This is a technique wherein required functions are not linked into a program until run time. This decreases the size of the program at the cost of having to load more than one package into memory. This technique is extensively used by Microsoft operating systems and software written to run on Microsoft operating systems.

DMA
Direct Memory Access.

DRAM
Dynamic Random Access Memory

DSP
Digital Signal Processor

Dynamic RAM
See DRAM

EEPROM
Electrically Erasable Programmable Read Only Memory. Sometimes written E²PROM.

FITS
Flexible Image Transport System. A file transfer format extensively used in astronomical and astrophysical work.

Flash memory
Nonvolatile memory which can be erased in situ and then reprogrammed, also in situ, using normal voltages.

FPGA
Field Programmable Gate Array. A gate array that can be programmed and/or reprogrammed by the end user, often under program control.

FWHM
Full width, half maximum

GUI

Graphic User Interface

Harvard Architecture

A computer technology wherein the program memory and the data memory are separate data spaces.

Indium-bump technology

A technique for mating two wafers. On one or both the wafers the intended connections have indium deposited on them. The mating faces are then aligned and brought together under pressure. If all goes well the indium will deform and form good electrical connections. Also known as “Flip-Chip” technology.

IR Array

A photosensitive CCD-like array designed to be sensitive to various wavelengths or bands of infra-red light.

ISR

Interrupt Service Routine.

Masked memory

A type of ROM in which the information is loaded during construction of the wafer by means of a deposition mask. This is an obsolete technology.

Mezzanine board

A circuit board mounted over another circuit board in order to gain board space (real estate) in a tight environment.

NASA

National Aeronautics and Space Agency. The United States Space Agency.

NMI

Non-Maskable Interrupt. An interrupt which can not be disabled.

Orthogonal Instruction Set

An instruction set where all instructions are available for all register combinations (or at least the register combinations that make sense).

OTP
One-Time Programmable.

Packed BCD
BCD digits, which take 4 bits each, packed two to a byte.

Page-mode
A type of flash memory where erasure is done a page at a time.

PAH
Poly-aromatic hydrocarbon.

PID
A control loop. The acronym means Proportional-Integral-Differential.

Platform
A computer, or microprocessor with all support systems: a complete entity.

PLD
Programmable Logic Device. A gate array.

PROM
Programmable Read Only Memory

Pseudo-ops
Computer instructions which look like machine language but for which there is no hardware to execute them. An interpreter is required for execution. This is a technique frequently used when putting together sequences of complex functions for which there is no direct hardware counterpart and where conservation of program space is more important than raw execution speed.

RAM
Random Access Memory

RAO
Rothney Astrophysical Observatory. The observatory operated by the Physics and Astronomy Department of the University of Calgary.

RCA
Radio Corporation of America.

RISC
Reduced Instruction Set Computer. A microprocessor optimized for speed of execution and/or minimum die size. These devices have small, non-orthogonal instruction sets but they execute at high speed.

RMM
Read Mostly Memory. See Flash Memory

ROM
Read Only Memory

RTD
A type of temperature sensor.

RTI
Return from interrupt. Since an interrupt is an asynchronous event, The return command must perform a complete context restore. Usually this consists of restoring selected registers before reloading the program counter with the address that the processor was executing from when the interrupt occurred.

Scratchpad memory
A small amount of RAM, usually located on-chip in a microcontroller. Since it is on-chip, access times are short.

SCSI
Small Computer System Interface. A computer interconnect standard.

SDRAM
Synchronous DRAM. A type of dynamic ram in which the refresh has been synchronized with the access to increase the speed of the part. These parts have become competitive with static RAM

Software Package
A set of software working for a specific purpose.

SRAM

See Static Ram

Static RAM

Random access memory in which the data is present, without refresh, for as long as power is applied. It is characterised by high cost, high speed, and low density.

UART

Universal Asynchronous Receiver-Transmitter. A serial device which does not require a clock signal from the device it is communicating with. It derives the clock from the data stream.

UVPROM

Ultra Violet Programmable Read Only Memory. A type of field programmable ROM which can be erased by the application of UV light.

Von Neumann Architecture

A computer architecture in which the data memory and the program memory utilize the same address space.

VME

A digital backplane standard originally developed by Motorola but now widely used. It is characterized by multiple address spaces and high speed. The most common VME card formats are 3U and 6U. A 3U card has one VME connector only. A 6U card has two VME connectors, with the second connector an auxiliary bus such as VXI, VXD, or VSE.. The actual VME standard is a five volt digital standard. The auxiliary bus may be either digital or analog.

VXD

- (1) An auxiliary digital bus used in a 6U VME backplane.
- (2) A device driver written for Microsoft Windows 9x or later operating system.

EPIGRAPH

Thy merchants chase the morning down the sea,
Their topmasts gilt by sunset, tho' their sails be whipped to rags.
Who raced the wind around the world come reeling home again.
With ivory, apes, and peacocks loaded, memories and brags.
To sell for this high profit:
Knowing fully they are men.

Poul Anderson

CHAPTER ONE
INTRODUCTION

The Moon! The Moon is to blame!

Drug addled cry frequently heard in Golden Gate Park, San Francisco, late 1960's.

Ah, yes. Must be the moon.

And reply.

1.0 INTRODUCTION

This thesis concerns itself with the design and construction of an infra-red camera built around a Rockwell TCM-1000C infra-red array, and intended for astronomical observation at the Rothney Astrophysical Observatory (RAO) at the University of Calgary. It is not a thesis concerning the science which can be done with such a camera, rather it is a chronicle of the engineering effort required to assemble such a device from disparate pieces.

As an example, the controller for the camera was obtained from Infrared Laboratories (IR Labs) in Tucson, Arizona, as was the dewar in which IR array is mounted. Some electronics for the array are derived from an original Rockwell design whereas other electronics were designed and built at the University of Calgary.

Some thousands of lines of C language were written specifically for this application at the University of Calgary, as well as approximately 1000 lines of assembler for the controller, not to mention an approximate 750 kilobyte (source) library downloaded from NASA and modified for the application.

The resulting instrument has utility in its own right as a scientific instrument, and is intended to be used as such. However, in many ways it is a pilot project, in that it demonstrates that the University of Calgary has the capability to design and deploy such instruments, and will serve as the springboard for the design and construction of infrared cameras with extended capabilities, including higher resolution, extended sensitivity range, and automated modes of operation.

Due to the fact that this thesis is more of an engineering paper describing an instrument than a scientific paper, it is divided into two major sections. These two sections are about the hardware design (chapters 2, 3 and 4), and the software design (chapters 5, 6 and 7). The hardware section consists of one chapter discussing general principles of data acquisition systems, followed by a chapter on the mechanical design of the IR camera, and by a chapter on the electronics in the camera. The software section consists of one chapter discussing general principles of real-time operating systems, followed by one chapter describing the real-time system running on the DSP in the IR Labs controller, and one chapter describing, on a high level, the host program running on the PC class computer and driving the camera.

The 1.8 metre telescope at the RAO is equipped with an IR detector capable of quite good photometric measurements, but only of point objects. Extended bodies cannot be investigated in detail with this detector for several reasons, including that as a point detector it was never designed for the investigation of extended bodies. Any attempt to obtain data concerning an extended body will result in data from an area which is the equivalent of a single pixel and from which all spatial information¹ has been lost. This is not satisfactory.

Four possible applications² for an imaging array at the RAO come to mind. The first,

¹

You could image with the InSb detector, but this would require exposing one pixel (equivalent) at a time. Not only is this not possible given the present state of the ARCT, but it would be painfully slow.

²

There are many more than four, but we shall discuss only these.

which is a photometric application and which can be done fairly simply, is to observe Seyfert galaxies³ and then to make isothermal maps. This would entail observations with

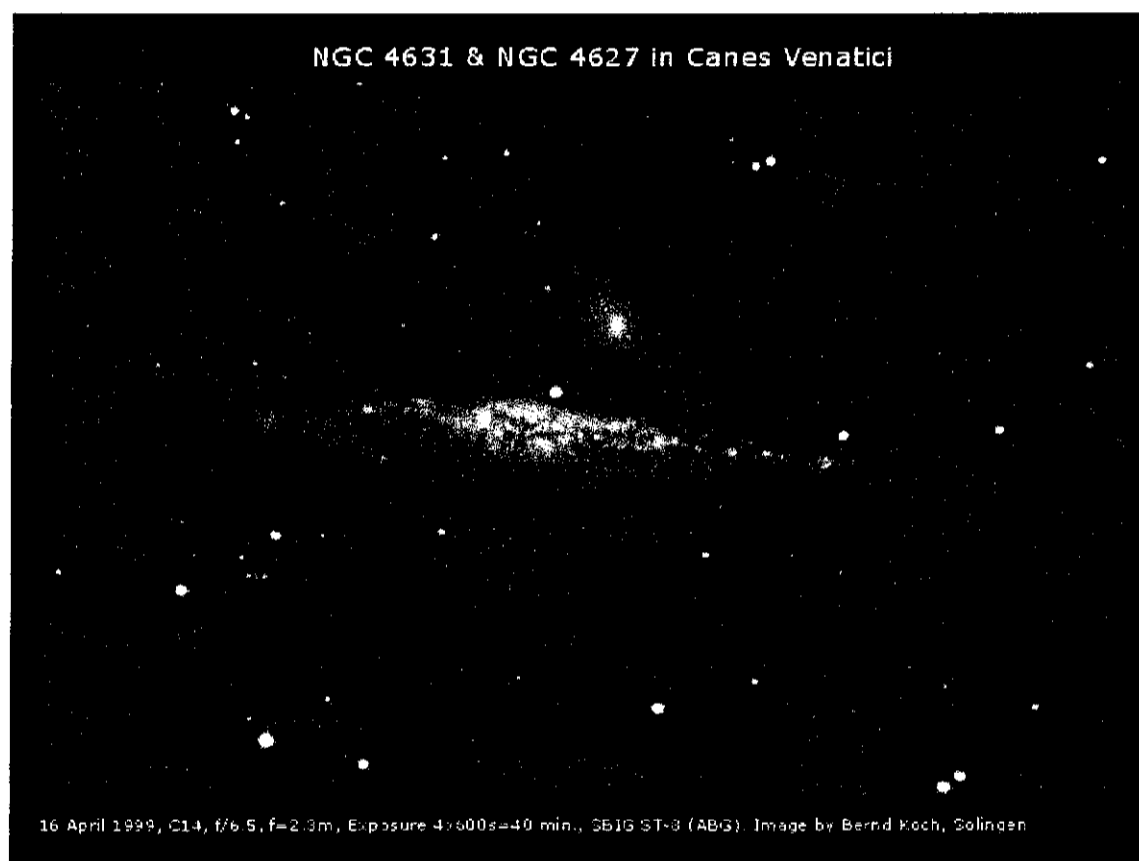


Figure 1-1. NGC 4631 in Visible Light.

various filters, the construction of pointwise colour value plots, and possibly the correlation of IR photometric data with data from other sources and for other portions of the spectrum.

The second application which comes to mind is the observation of planetary nebulae.

3

Such as NGC 4631, shown here as Figure 1-1, in a photo taken from the collection of Bernd Koch, at the University of Bonn, Germany. (<http://www.astrofoto.com/astro/>).

The difficulty of studying these objects with a point detector is that a point detector provides at best data only from the equivalent of one pixel, when it is the detail of the image which is of the essence. With a planetary nebula, there is an outer shell of gas and dust and a hot point object in the centre which is the star. The shell is formed when the older, low-velocity stellar wind from the star is impacted from behind by a high-velocity wind, also from the star. An examination of the temperature of the gas, and in particular an examination of the shock zone, could reveal details concerning the velocity of the high-speed gas and dust and thereby details concerning the star within.

The next application is the observation of protoplanetary disks. Areas such as the Trapezium⁴ have dust clouds of various optical depths. Protostars are forming in these regions, however even those protostars which have already achieved hydrogen ignition are difficult to see in the optical portion of the spectrum due to the dust. They are quite visible in the infra red, but an imaging camera is required to resolve the protostar from the obscuring dust and gas.

As remarked in the paper cited above, an efficient technique for performing a census of circumstellar disks around young stars is to obtain infrared imaging surveys of the stellar clusters in which these stars are forming. This information can then be used as a preliminary step in determining the frequency of planetary systems in the Milky Way. Since

4

Lada, Muench, Haisch, Lada, Alves, Tollestrup and Willner, Reference 22. The Trapezium is a hotbed of circumstellar disks, candidate protostars and dust clouds. Imaging in J, H, K, and L bands indicates a high level of detail.

circumstellar disks are bright IR emitters and have an IR excess over that of the photospheres of young stars, they are particularly amenable to measurement with a near-IR imaging camera such as the one which is the topic of this thesis. Colour-colour diagrams can then be constructed, using data obtained from the camera and from other imaging sources, and these diagrams are known to be useful in the identification of objects displaying infrared excesses, and in identifying candidate circumstellar disks.

There is also evidence which indicates that the frequency of sources exhibiting excess in the J, H, and K bands decreases as a function of the cluster age, with time frames of a few million years. This may be associated with constraints on the lifetimes of circumstellar disks and the time frame available for planetary formation.

Since protostars are thought to display large L-band excesses and colours indicating that they are embedded in dust clouds, the camera is well suited to the observation of these objects. However, as the above cited paper points out, due to background variations in the L-band⁵ and other problems with observation, the quality of the data (taken 1997) was poor. Though further data was taken (in 1998) to augment the data obtained the previous year, the situation was still somewhat unsatisfactory, indicating the need for further observations. This, the IR camera can do.

An example of the type of infrared imaging which can be done is shown overleaf as Figure 1-2. This is a mosaic image of the Trapezium, from the Lada, et al. paper, showing

⁵

Reportedly due to window dust, instrument/telescope flexure, and detector anomalies, as well as significant L-band emission due to H II regions in the field of view.

the Trapezium as it appears in the L-band.



Figure 1-2. Trapezium⁶ in L-Band, from Lada, et al.

Interesting features which are pointed out in the paper include the bright bar seen in the southeast, and emission from dust at 3.3 micrometers, due to small PAH grains which

⁶

This is Figure 1b in the original paper, which in colour in that paper. It is reduced to grayscale in this reproduction for the purpose of printing.

are stimulated by UV radiation from the early stars in the cluster.

In 1994 Lada, et al⁷. showed that a technique was available which allowed the mapping of dust distributions in cold molecular clouds⁸. This technique used data obtained in near-IR imaging surveys, combining direct measurements of IR colour excess along narrow beams with sampling procedures. While this technique requires the observation of a great many stars behind the cloud being investigated, it does allow the derivation of the spatial distribution of extinction and the mapping of dust column densities across the cloud. In the 1994 paper they report having produced an extinction map with an angular resolution of 90 arc-seconds. They also report having found a positive linear correlation between the mean extinction and its measured dispersion, and were able to show that the data obtained is not compatible with cloud models having uniform extinction or with having many small, discrete, high-extinction areas.

In a later paper Lada, et al⁹ discuss the application of this technique to an investigation of the Northern Streamer portion of IC 5146. In this paper they discuss work done in 1995 at Kitt Peak National Observatory using the 2.1 metre reflector, with a NICMOS III¹⁰ array as the detector. They report that with this equipment they were able to

⁷

Lada, C. J., Lada, E. A., Clemons, D., & Bally, J. 1994, ApJ, 429, 694

⁸

Their test case was IC 5146.

⁹

Lada, C. J., Alves, J., Lada, E. A. 1999, ApJ 512, 250

¹⁰

The NICMOS III chip is a 256 by 256 HgCdTe IR array manufactured by Rockwell

obtain a FWHM spatial resolution of 1 pixel approximately equal to $1''.09^{11}$. With this, after the application of some spatial filtering and other data extraction techniques, they were able to obtain the first distance estimate to the Northern Streamer (approx 460 pc). They were also able to derive detailed contour maps of the equivalent visual extinction in the Northern Streamer (their Figure 4 reproduced here as Figure 1-3).

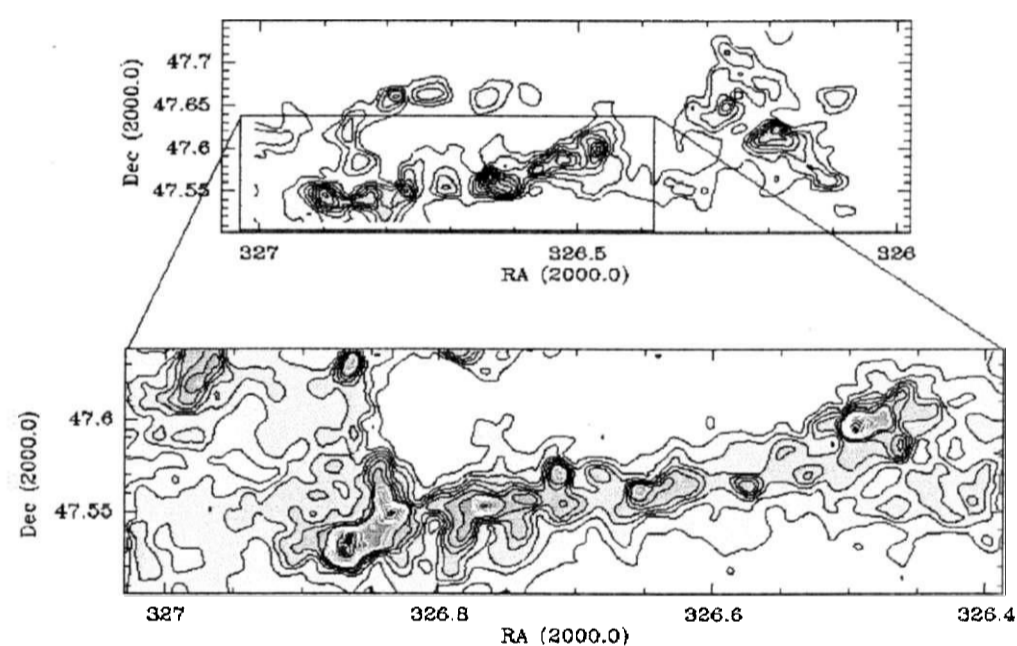


Figure 1-3. Contour Maps of Equivalent Visual Extinction, Northern Streamer

Scientific Corporation. It was replaced in 1996 by the PICNIC chip, which also is 256 by 256 and which has the same spectral range. The spectral response curve for the PICNIC is shown at the end of this chapter as Figure 1-5. While the NICMOS III and PICNIC detectors are descendants of the TCM-1000C they have a different spectral range than does the TCM-1000C. The spectral response curve for the TCM-1000C is shown at the end of this chapter as Figure 1-6, and is included for comparison.

11

In our case, with the ARCT at the RAO and an estimated field of view of 55 arc-seconds across 128 pixels, we are estimating a spatial resolution of $0.43''$ per pixel.

All the above applications are deep-field applications requiring¹² the 1.8 metre telescope at the RAO. However, there is a heliostat¹³ at the RAO, and with the aid of a suitable neutral-density filter it can be used for solar observations. Images such as that shown in Figure 1-4 (overleaf) are possible to obtain by means of this camera.

In the wild-eyed-idea department, there is a continuously variable infra-red filter available for the 2 to 4 micrometer range¹⁴. While this author has not seen the device and has no data on its operation, it is reported to be able to slice its operating range into approximately fifty bands¹⁵. If so, it might be possible to construct an infra-red camera using the Rockwell TCM-1000C (or another IR array), the continuously variable filter, and to use this, for example, to obtain information concerning the makeup of the gas and dust clouds surrounding protoplanetary disks as previously discussed. Preliminary discussions indicate that the existing dewar cannot house both the IR array and the continuously variable filter, and there is some concern about the optics required to use this filter with an imaging

¹²

The Cassegrain focus on the 41 cm telescope at the RAO is f/20, which is much too slow for the IR camera. It is therefore either the ARCT or the heliostat, if the IR camera is to be used at the RAO.

¹³

The heliostat has to be assembled and mounted, but all the pieces are on site, and the instrument has been used at the RAO before.

¹⁴

Personal communication, Dr. T. Alan Clark.

¹⁵

Personal communication, Dr. D. J. I. Fry.

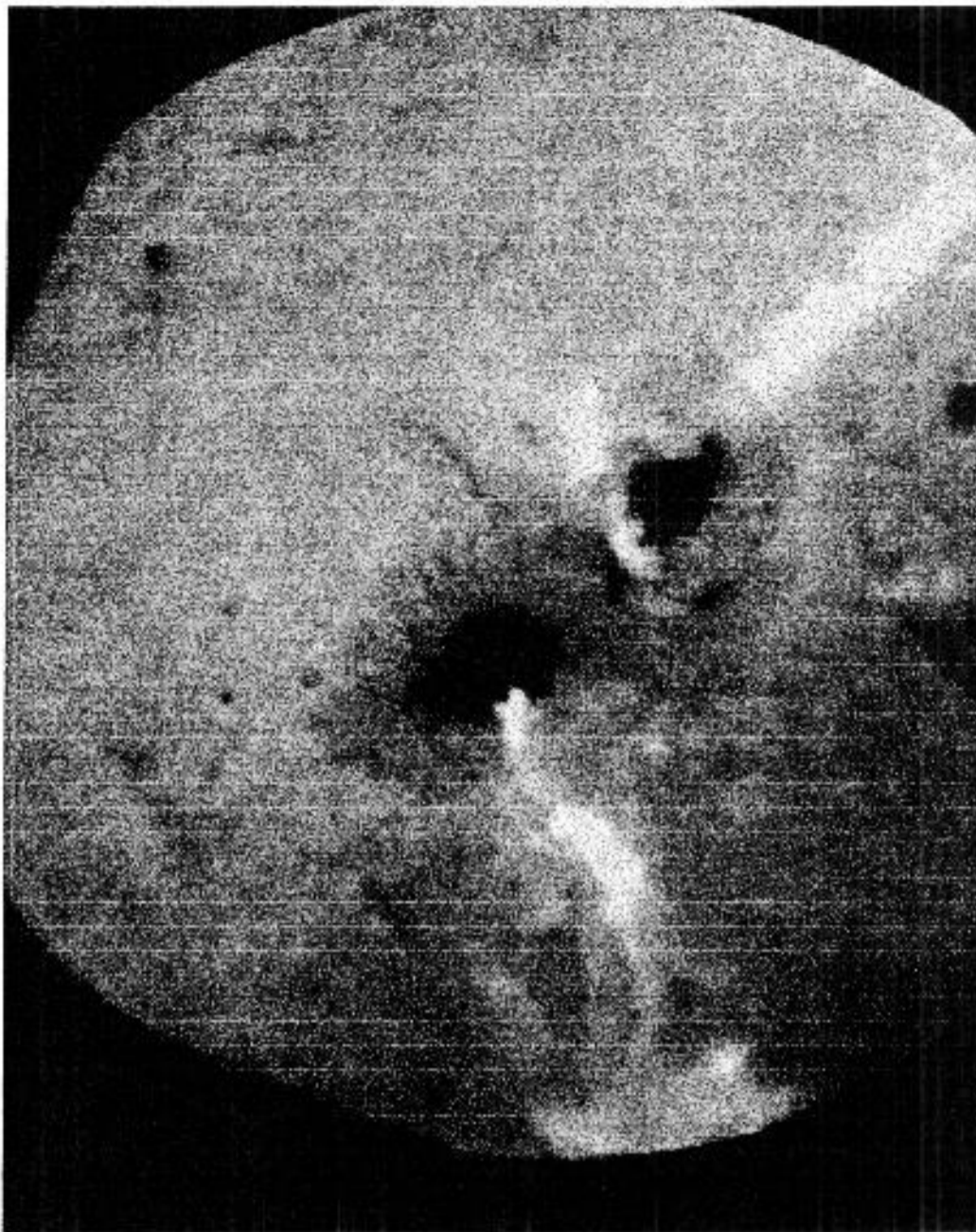


Figure 1-4. Solar flare at Near-IR Wavelengths, 6 September 2001.

Courtesy Dr. T. A. Clark.

detector¹⁶.

Another possibility is the concept of using an imaging array, a slit, a diffraction grating, and some optics to devise a synthetic-aperture imaging spectrometer¹⁷.

This would allow great precision for spectroscopic data while at the same time preserving spatial information. While this concept has not progressed beyond the back-of-the-envelope stage, it has been marked for further investigation and will likely result in a white paper.

¹⁶

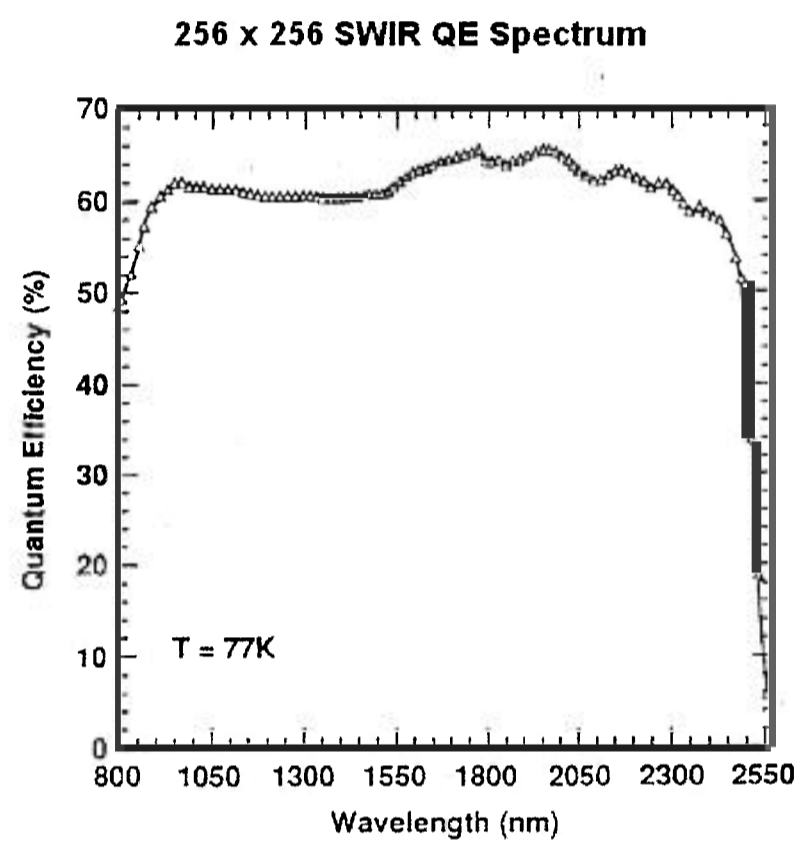
There are species in gas that have transitions in the IR (Kwok, Sun, "Physics and Chemistry of the Interstellar Medium", draft version, January 2000). If the filter bands are narrow enough that these transitions do not overlap, data concerning what is present and where it is present can be obtained. Assuming the filter resolution is good enough, it might be possible to use Doppler broadening and line shifts to get information concerning gas and dust temperature and velocity. However, as pointed out by Dr. D. J. I. Fry, the continuously variable filter is designed for a point detector, and the passband is a function of where on the filter the light hits it. Thus, it might not be suitable for a plane array detector, and in any case would require careful optics design to guarantee uniformity of response.

¹⁷

Personal communication, Dr. D. J. I. Fry. The idea is to scan the image using the slit, project the light onto the grating, and then project the resulting two dimensional entity onto the IR array. By then turning the scanner at right angle to the original scan direction and doing it again, it should be possible to obtain enough information to be able to back out images at wavelengths of your choice. To do this successfully would require great precision in the pointing of the telescope (see Chapter 8) and some degree of proficiency with tomographic techniques.

RSC Imaging Sensors**PICNIC**

Focal Plane Arrays, Electronics, and Camera Systems

Figure 1-5. PICNIC Spectral Response Curve¹⁸

¹⁸From Rockwell RSC website, <http://www.rsc.rockwell.com/imaging/picnic/256qe.html>.

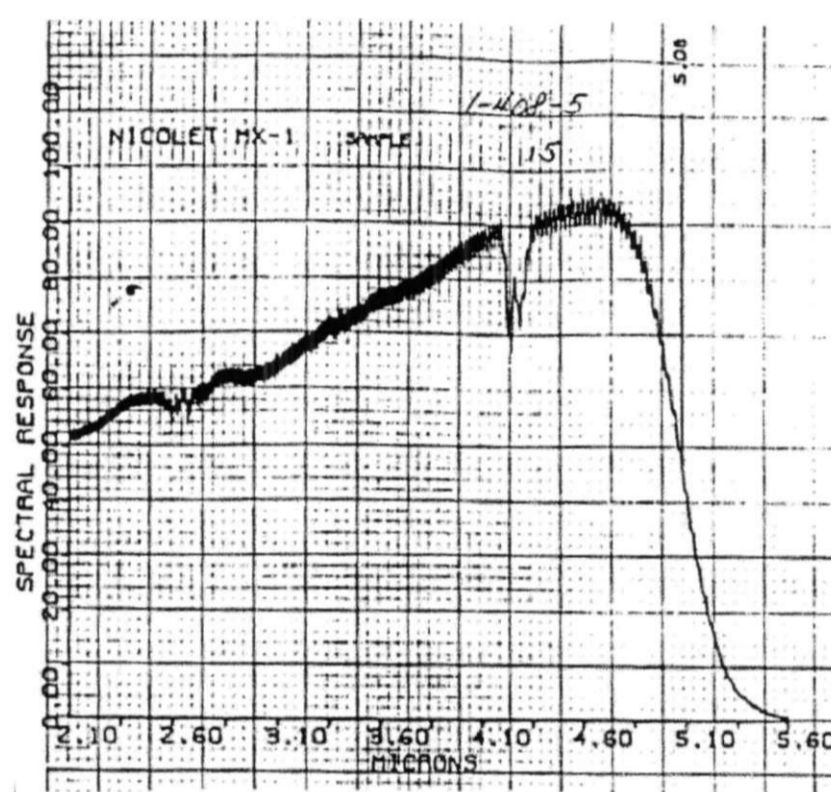


Figure 1-6 Relative Response as a Function of Wavelength for TCM-1000C IR Array¹⁹

¹⁹

From Rockwell engineering report SC87004.FR, Reference 53.

CHAPTER TWO
GENERAL CHARACTERISTICS OF DATA ACQUISITION SYSTEMS

Set the controls for the heart of the sun.

Roger Waters

CHAPTER 2.

GENERAL CHARACTERISTICS OF DATA ACQUISITION SYSTEMS

2.0 Introduction

In general terms, a data acquisition system can be considered to be a transducer linked to a recording device. The transducer can be as simple as a voltmeter, and the recording device can be as simple as a person with a timer, pencil, and paper. It is not the business of the data acquisition system to interpret the data, simply to acquire and record.

Good examples of basic data acquisition systems include chart recorders, recording meters, and strip chart recorders.

More complex data acquisition systems can automate the data recording function, releasing the person who would otherwise be engaged in recording the data displayed by the transducer. An advanced data acquisition system can be equipped not only with local storage capability but with remote reporting capability, using a variety of wired and wireless transmission technologies. However to perform remote reporting effectively requires that the data acquisition system have some degree of local intelligence in the form of a microprocessor or microcontroller, and to equip a data acquisition system with such a processor opens up a whole new level of capability not possible with simple recorders.

2.1 Requirements for an Advanced Data Acquisition System

An advanced data acquisition system will have one or more of the following features: communication to a host; remote operation capability; process control capability; stored

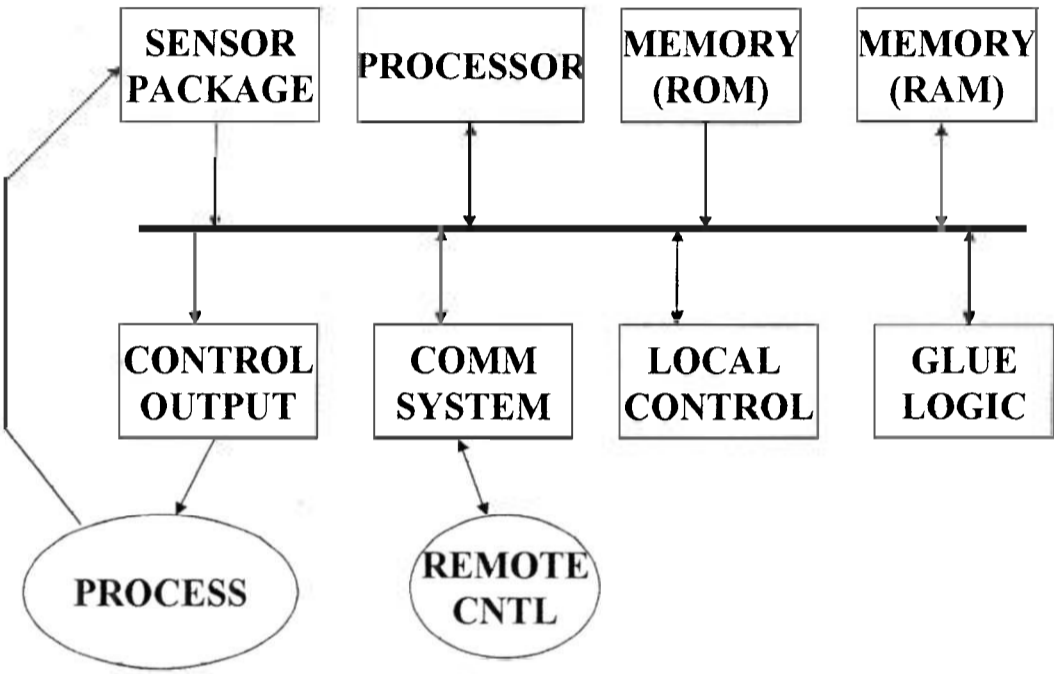


Figure 2-1. Block Diagram Of Data Acquisition System

program update capability; and hardware reconfigurability. It is not necessary for a data acquisition system to be powered from a mains supply as advances in CMOS technology¹ have made available devices with very low power consumption requirements. Data acquisition systems constructed with this technology can be battery powered, and can be combined with wireless reporting capability to produce an instrumentation package that can be placed in a remote location. Often such packages, especially if mass-produced or if adapted from mass-produced packages, can be produced cheaply enough that they can be used in situations where recovery of the package is not expected to occur.

For example, the APTEC-NRC DRM-300/600 series radiation meters. These are handheld recording ratemeters built around an RCA COSMAC-1800 series microprocessor, which are battery powered, and have many of the features described above.

2.2 Elements of an Advanced Data Acquisition System

As remarked above and as illustrated in Figure 1-1, an advanced data acquisition system is equipped with an embedded microprocessor or microcontroller. It will have local data storage capability but may also have data transmission capability, and will have stored program capability. It may have the capability to receive program downloads from a host, and advanced systems may have the capability to download hardware reconfigurations² and change their hardware characteristics on command. Frequently, data acquisition systems have process control capability³ and can run a single-point control loop or a more complex control algorithm independent of the host computer.

2.2.1 Microprocessor Options

There are numerous choices for the microprocessor, and the selection should be made on the basis of unit cost, required flexibility, expected production volume, required performance, and projected availability of the component. Some classes of microprocessor/microcontroller are more difficult to design with, and are more difficult to

²

Systems equipped with SRAM-based FPGA logic can do this.

³

Omega Corporation, for example, has an extensive line of rack-mountable single-point PID and bang-bang controllers. These controllers are microprocessor based and are equipped with serial ports, (RS-232, RS-422/423, RS-485) so that they can be tied to a host. Although they are designed primarily for temperature control using thermocouple, thermistor or RTD, some will accept a voltage input. These last can be re-programmed from the host to operate as general-purpose single-point controllers running a self-tuning PID loop and reporting to the host via the serial link.

program for, than others, an aspect which will drive up the engineering cost, but the unit cost, especially in volume, may be such that for large production volumes it is cost effective to utilize such a device. Generally, the Harvard architectures are more difficult to

TYPES OF MICROPROCESSORS

- **SINGLE ADDRESS SPACE FOR PROGRAM AND DATA (VON NEUMANN ARCHITECTURE)**
- **MULTIPLE ADDRESS SPACES, SEPARATE FOR PROGRAM AND DATA (HARVARD ARCHITECTURE)**
- **SINGLE-BUS STRUCTURE, WITH OR WITHOUT INTEGRAL DMA**
- **MULTIPLE-BUS STRUCTURE, WITH INTEGRAL CONCURRENT OPERATION DMA**
- **MULTIPORT PINOUT**

Figure 2-2. Types Of Microprocessors

program for, but they usually have a cost advantage and are useful in applications where the code is relatively static over time. This class of processors may also have issues concerning differences in word size⁴ between the data memory and the program memory which make

⁴The Analog Devices ADSP-2100 family, for example, has this characteristic.

downloading new code from a host difficult. If the data memory and the program memory are brought off-chip through separate ports⁵, or if there is autonomous DMA capability for both data memory and program memory coupled with sufficient internal scratchpad, then Harvard architecture microprocessors will enjoy a speed advantage over Von Neumann machines. Examples of cost/feature tradeoff for various microprocessors

ARCHITECTURE	DEVICE	COST	CLOCK SPEED
VON NEUMANN	INTEL 80C51	CHEAP (±\$1.00 EA)	SLOW (1-5 MHZ)
HARVARD	ANALOG DEVICES 2800	MODERATE (±\$2.50 EA)	MODERATE (12 MHZ)
SINGLE-BUS WITH DMA	TI 320C16	MODERATE (±\$10.00 EA)	MODERATE (20-33 MHZ)
MULTIPLE BUS WITH DMA	TI 320C30/40/50	HIGH (±\$30-150 EA)	FAST (33-100 MHZ)
MULTIPORT PINOUT	TI 320C30	HIGH (±\$30-50 EA)	FAST (33-100 MHZ)

Table 2-1. Examples of Microprocessor Architecture

⁵

The Motorola DSP56000 family of processors uses a Harvard architecture with two data memories and one program memory. There are reasonable but not overly generous on-board SRAM scratchpads for each memory, but then all memories are accessed through a single bus structure. From an electrical engineering point of view this prevents an “explosion of pins” with associated fabrication problems and also holds the unit cost down, however it has the effect of driving off-chip access times up. This family of processors also is not orthogonal in the sense that the instruction set is register-specific. These problems among others make writing high-performance code difficult at best.

are shown in Table 2-1.

2.2.2 Memory Options

There are two types of memory to be considered. The first type of memory to be considered is read/write memory, commonly known as RAM. The second type of memory to be considered is read-only memory, commonly known as ROM.

2.2.2.1 RAM

Read/write memory is available in two major types. First, there is static RAM (SRAM), which uses a full six-transistor flip-flop structure⁶ for each bit. This type of memory is quite fast⁷ but due to the cell structure utilized on the chip it is relatively low density. It is therefore best suited to smaller systems⁸ or as cache memory in larger systems, coupled to mass memory via an explicit memory manager.

The other major type of RAM is dynamic RAM (DRAM), which uses only one transistor and a capacitor cell for each bit. The great advantage of this type of memory is that it can be made very cheaply and has very high densities⁹ but the disadvantage is that

⁶

Micron Semiconductor

⁷

3 nanosecond access times are common, 5 nanosecond access time components are becoming very reasonably priced.

⁸E.g., systems with a 16-bit address space.

⁹64 M by 1 bit memories in a single package are common.

dynamic RAM must be periodically refreshed. Aside from the power requirement for refreshing the memory, there are the issues of the hardware required for the refresh, the requirement for an explicit memory manager to handle the DRAM and to keep the static cache and the DRAM synchronized, and finally the access time for the DRAM¹⁰.

2.2.2.2 ROM

Read only memory (ROM) has come a long way since its inception. A few years ago, ROM was available from the vendor as only masked memory, and the vendor was happy to sell you a thousand pieces with your program hardwired into the memory. It was not changeable, and what you got was what you got.

Field programmable memory (PROM) changed all that. While this type of memory could be programmed only once, it could be programmed by the user in the field and it was not necessary to provide a bitmap to the chip vendor and to purchase a large number of pieces in order to get a ROM. As the technology improved, ultraviolet erasable stored-charge devices (UVEPROM) became available, and it was possible and very affordable to purchase a device that could be programmed, erased, and then reprogrammed.

10

Ten years ago (1990) 60-nanosecond access times for DRAM were considered blindingly fast. At the turn of the millennium, with synchronous DRAM (SDRAM), 7 nanosecond access times are available, though you will pay for it.

2.2.2.3 RMM

However while the UVPR0M was the dominant technology, a new technology, referred to as Read-Mostly Memory, or RMM was emerging. These were devices with limited storage and a limited number of erase and write cycles, but they could be erased electrically and written to electrically, and data stored in them was not volatile. Digital designs¹¹ in the late 1980's and early 1990's used these devices to store configuration information unique to each unit and which could not be determined at design time, so that this information was available immediately on boot.

2.2.2.4 Flash Memory

An outgrowth of RMM is flash memory¹². Flash memory is a non-volatile mass storage medium with the attribute of erasability while in-circuit and either an unlimited or a very high number of erase-and-write cycles¹³. Modern devices have advanced features allowing erasure of only selected portions of the memory¹⁴, thus allowing critical data to be protected against inadvertent erasure during reloading of the device. This attribute makes

¹¹

Such as the APTEC-NRC ADM-300 ratemeter.

¹²

Such as the Intel 28F010 memory and similar devices.

¹³

Advanced Micro Devices, with their Standard™ family of flash memory are guaranteeing one million write cycles per sector and 20 years of data retention. Devices in this family are available for 1.8 volt, 3.0 volt, and 5.0 volt logic.

¹⁴

Specifically boot-block and page-mode devices

these devices ideal for storage of firmware in systems which will be deployed in remote or inaccessible locations and which must also deal with firmware updates from a remote host.

2.2.3 Reconfigurable Logic

With the advent of complex programmable logic devices (CPLDs) it became possible to purchase a generic high-density logic device¹⁵, include it in a design, and then program it in-situ after the design was built. This technology obviated the need for ASICs in all but the most specialized designs while still allowing the advantages inherent in cell-based complex logic. The problem with a CPLD however is that it can be programmed only once. While the fact that it is programmable at all allows great flexibility in the utilization of the device, the fact that it is programmable only once means that the structure impressed on the CPLD by the programming cannot be updated.

There are two ways out of this dilemma. The first way is to use flash memory¹⁶ instead of an OTP¹⁷ configuration. Then, the device can be reprogrammed, although not usually in the field. However, the configuration data is non-volatile and is available on power-up.

¹⁵

E.g., a 22V10 or 16L8

¹⁶

A good example of this is a XILINX 5200 series CPLD or one of the ATMEL flash memory based offerings.

¹⁷

One Time Program

The second way is to employ a static RAM (SRAM) based cell structure¹⁸ in the CPLD. This results in a field programmable gate array (FPGA). The disadvantages to this approach are that, since the configuration is in SRAM, it is volatile and is not available at power-up, and also that at least one more component¹⁹ must be included in the design to hold the configuration data and to make it available to the FPGA on boot. The real advantage, at least from the system designer's point of view, comes with the realization that while the vendor is quite willing to sell serial memories which will mate seamlessly with their FPGAs, the FPGA will also accept byte-wide programming. This means that the configuration port for the FPGA can be put on the data bus for the microprocessor, and although there must be some non-volatile logic holding everything together during boot, the FPGA can be programmed after microprocessor boot and therefore under the control of the microprocessor. The great advantage of this is that it is now possible to reconfigure and update the operation of the data acquisition system while it is in operation, and in the field. It is further possible to reconfigure or update the data acquisition system remotely via

¹⁸

The most successful SRAM based devices are the XC4000 family FPGAs and their derivatives, available from XILINX. These devices are not cheap in themselves, however they do offer some extremely high logic densities, and in some of the larger parts it is possible to stuff an entire 8051 core into the part. Since this same part is available with 2 nanosecond or better delay times, a very powerful system can be built at low cost.

¹⁹

XILINX offers a X17Cxx line of OTP memories which are expressly designed to mate with their FPGAs. These are serial-download devices which will download their contents to the FPGA under the control of a hardwired FPGA protocol. ATMEL sells a similar device, except that theirs is based on flash memory and is therefore reconfigurable.

the data link, and to store the new hardware configuration in flash memory for later use.

2.2.6 Building The Platform

Referring once again to Figure 2-1, we see that by use of the building blocks described above it is possible to design a data acquisition system with an embedded microprocessor, capable of running autonomously if required but also equipped with (an optional) communication subsystem, and capable of great flexibility in terms of both hardware and firmware configuration. With all this capability available however, the precise design will depend on the mission statement for the data acquisition system, coupled with such other factors as size of production run, availability of components over the projected lifetime of the system, and any management-imposed constraints.

2.3 IR Arrays and Charge-Coupled Devices

At first glance the difference between a charge-coupled device and an IR array is somewhat artificial in that they both employ a photovoltaic sensitive element coupled to a storage well and readout electronics. The major difference between the two is that generally a CCD is oriented toward the visible portion of the spectrum whereas an IR array is not sensitive (or at least not deliberately so) in the visible portion of the spectrum but rather is designed to operate in various portions of the 1 to 10 micrometre region of the spectrum. Another difference, though a difference in implementation rather than a difference in basic technology, is that while CCDs generally operate at temperatures ranging

from ambient to $-100\text{ }^{\circ}\text{C}$ it is always necessary to cool an IR array and the enclosure in which it is mounted in order to cut down on noise and extraneous current. This last means that the design of an IR array is somewhat different from that of a CCD and that the materials used in the manufacture must be materials that will have the desired electrical properties at the operating temperature, and independent of what their electrical properties are at room temperature.

The fabrication of IR arrays is also somewhat different from that of CCDs. A CCD usually is a high-production-volume device, and if a designer wants to design a system using a CCD she can find a plethora of devices available off-the-shelf and there is little need to design her own. However, the market for IR arrays is somewhat more restricted and the operating conditions are somewhat more stringent, so it is not uncommon for such a device to be fabricated as two components (one being the sensitive element and the other being the support electronics) and then for the two components to be mated using something akin to indium-bump fusing technology²⁰. The combination of low production volume for IR arrays,

²⁰

The RENA chip (Nova Research, Riverside, California) was fabricated this way. In this case the region of interest in the spectrum was in the MeV range [for an instrument to detect gamma-ray bursters], and the sensitive element was a material appropriate for this portion of the spectrum. The support portion of the device was an ASIC fabricated as an NMOS wafer and the sensitive element was mated to the ASIC using indium bump.

There are several vendors which will accept academic and/or prototype designs and run them through a low-volume silicon foundry. An example is the 0.35 micron MOSIS process available to and in use by the University of Alberta Microelectronics Centre, among others.

combined with somewhat lower yields for a more complex process results in detectors which are very expensive when compared to the cost of the mass-market CCD.

However, the CCD is itself under pressure from CMOS image sensors. While these devices have somewhat poorer performance than CCDs, they can be fabricated using normal CMOS technology, which means that any silicon foundry can make them, and quite cheaply. Further, since they are CMOS, their power consumption is much lower than that of a CCD. Finally, they can be designed so that each pixel can be accessed separately rather than through a shift register setup, which makes for ease of design of the support electronics. They are in the classic sense cheaper than anything better and good enough to do the job for most applications.

2.3.1 Operating Considerations for an IR Array

The quantum efficiency of an IR Array is quite high - figures of 85 percent are not uncommon²¹. However, they must be operated at cryogenic temperatures, which means that a dewar with all the associated plumbing must be supplied. Further, while they can be fairly

²¹

Jobin Yvon, Inc. offers, in their IGA3000 line of near infrared array detectors, devices with spectral coverage of 0.8 μm to 1.7 μm as linear arrays of 128, 256, or 512 pixels. They quote 85 percent quantum efficiency for these detectors. The telephone number is 732-549-5125.

large²² the number of available output ports is limited²³ and the output ports are analog. This requires that support electronics be supplied in order to buffer and amplify the output, and to convert the output from analog to digital. This also requires that a relatively complex set of timing and control signals be provided to the array. At a minimum, a frame line, a synch line, and a clock must be provided²⁴.

There are also voltage and power considerations. Due to the nature of the technology there are several bias voltages driving the photodetector section. These must be supplied from stable, clean sources, external to the dewar. The readout electronics also do not use standard logic levels, requiring level shifters in the support electronics.

The method of output of either an IR array or a CCD is referred to as the Bucket-Brigade technique. In this technique, the charge accumulated for each pixel is successively shifted to the next pixel until it arrives at the output line. It then enters an integrator where it is developed as a voltage. This voltage is then presented to an ADC, and the output of the ADC is stored as the value for that pixel. The array incorporates a shift register so that each time the synch line is strobed it will step to the next line in the array, allowing the entire array to be read out. The array shifts charges on the leading edge of the clock and the

²²

256 by 256 arrays are readily available, and these can be and are assembled into larger mosaics.

²³

The IR Labs NICMOS array has four output ports, the Rockwell TCM-1000C has two.

²⁴

e.g., the Rockwell TCM-1000C

output is valid on the trailing edge of the clock.

All of this then requires a controller to handle the IR array, and this controller is in the classic sense a data acquisition system. In our case the controller is an IR Labs purpose-built array controller based on a Motorola DSP56002 microprocessor, resident in a 3U VME chassis²⁵ and communicating with a host computer via a parallel interface²⁶.

2.4 IR Labs Controller

The IR Labs controller is housed in a six-slot 3U VME chassis with an on-board power supply which supplies digital power as well as ± 10 volt analog supplies. There are three cards in the controller. These cards are the Timing Card, the Clock Generator Card, and the Analog Input Card.

2.4.1 Timing Card

The timing card has on it the Motorola DSP56002 and support for the DSP, including the UVROM which holds the program loaded into the microprocessor at boot. There are several programmable logic devices on the card, however the configuration of these devices is not accessible to the DSP and they are used to achieve a high level of integration of the glue logic on the board. They are non-volatile. The timing card has a low-speed serial

²⁵

This is not a standard 3U VME backplane. All lines have been redefined for the purpose.

²⁶

This is a SCSI interface with lines redefined for the purpose. IR Labs has supplanted this technology with a fibre-optic link between controller and host in later designs.

channel to the host and a high-speed parallel channel to the host.

The firmware in the timing card is resident in a UVROM²⁷ and is loaded into an on-board RAM program memory in the DSP on boot²⁸. The timing card generates all timing and control signals for the Rockwell array.

2.4.2 Clock Generator Card

The clock generator card has a set of D/A converters for each output line and a set of analog switches for each output line. The timing board can set the outputs of the D/A converters, and then can toggle between the voltage outputs of two D/A converters to generate a digital signal. Because the clock generator card is organized into groups of lines it is possible to generate either single-ended or differential digital signals, depending on what is required.

2.4.3 Analog Input Card

The analog input card is a two channel card. Each channel consists of an analog to

²⁷

In the incarnation used at the University of Calgary. The card will also support EEPROMs and if these devices are used then the timing card can download and store programs from the host as required.

²⁸

There are several modes of operation for the DSP56002. In the mode of operation implemented in the IR Labs controller, an address overflow of the program memory will cause the DSP to reach out through its external port to a static RAM on the card. This allows programs larger than the size of the scratchpad memory to be run on the DSP. The problem is, both data memories also use that same port, and there are bus contention problems and timing issues which result when off-chip memory is accessed.

digital converter for conversion of the signal from the IR array or CCD, with upstream integrator, variable gain amplifier, and offset voltage adjust. This card also is equipped with D/A converters²⁹ to generate the bias voltages to drive the array. The output from the A/D converters is 16 bits wide and is directly transferred to the host via the high-speed parallel link under the control of a PLD. The host then stores the data in a buffer using DMA.

2.5 Host

The host for this system is a Pentium III class personal computer running under Windows 98 and equipped with a Spectral Instruments® Model 1862 Parallel Interface card. This card is a PCI card and forms the nether end of the link with the IR Labs controller. The host runs custom software to drive the controller and the Rockwell TCM-1000C via the IR Labs controller.

2.6 Details of System.

The electrical and mechanical features of the IR camera will be discussed in Chapters 3 and 4 of this thesis. The structure of the host software running on the PC, and the firmware running on the Motorola DSP56002 microprocessor will be described in detail in Chapters 6 and 7.

²⁹

Not used at the University of Calgary with the Rockwell TCM-1000C

CHAPTER THREE
IR CAMERA ELECTRONICS AND CABLING

The fault, dear Brutus, lies not in the stars but in ourselves,
that we are men.

Wm. Shakespeare

CHAPTER 3.

IR CAMERA ELECTRONICS AND CABLING

3.0 Introduction

There are four major electronic subsystems in the IR camera. These subsystems are the dewar electronics, including the Rockwell¹ TCM-1000C array and support electronics; the RAO designed preamplifier and support electronics; the IR Labs² array controller, and the personal computer that drives the camera. The first three subsystems will be discussed in this chapter, and the Spectral Instruments³ 1826 card in the computer will be discussed as well. The balance of the computer will be ignored except to note that it is an Intel Pentium III system running under Microsoft Windows 98.

3.1 Dewar Electronics

The dewar electronics subsystem is composed of the Rockwell TCM-1000C array, the support package for the array, and a Honeywell microswitch. There are also two Guardian⁴ solenoids in the dewar to work the shutter, as shown in Chapter 4.

¹

Rockwell International Science Center, now Rockwell Scientific Company, LLC. P.O. Box 1085, Thousand Oaks, CA., 91358, USA. 805-373-4545

²

Infrared Laboratories, 1808 E. 17th St, Tucson, AZ, 85710-6505, USA. 520-622-7074.

³

Spectral Instruments, 1802 West Grant Road, Suite 110, Tucson, AZ, 85745, USA. 520-884-8821.

⁴

Part number LT6X12C24D

3.1.1 Rockwell TCM-1000C Array.

The Rockwell TCM-1000C array is a prototype⁵ manufactured for IR Labs sometime around 1987 by mating a PACE-1 HgCdTe-on sapphire detector array to a TCM-1000C readout circuit using indium-bump technology. The array is a 128 by 128 pixel format with a 60 micrometre square pixel size. It is sensitive to short wavelength infrared in the range of 2.0 to 5.0 μm with about a 40 percent spectral response at the short end, peak sensitivity at about 4.6 μm and rapidly falling off thereafter. The response curve from the Rockwell documentation is reproduced as Figure 3-1, below.

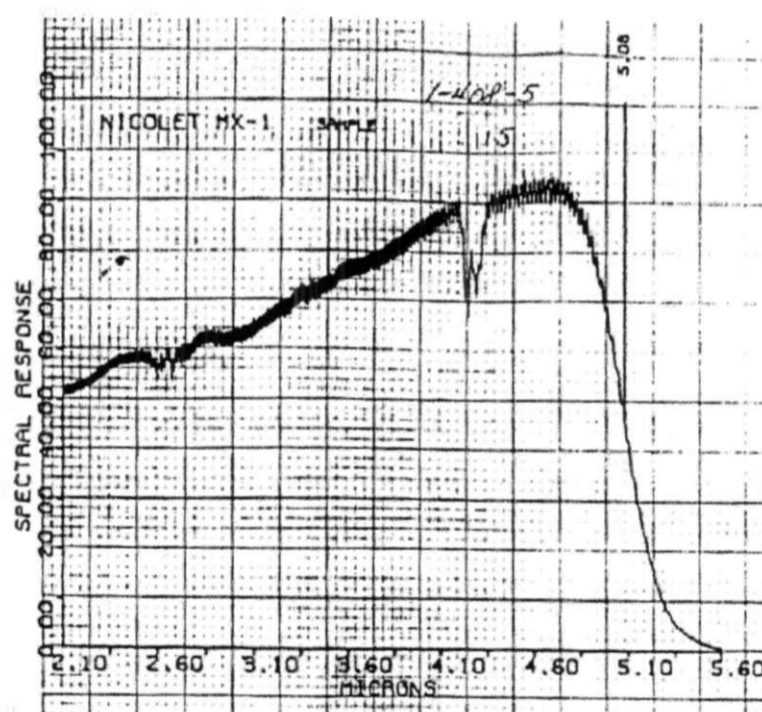


Figure 3-1. Relative Response as a Function of Wavelength for TCM-1000C IR Array

⁵ IR Labs purchase order 7597 as reported in Rockwell report SC87004.FR, Reference 53.

The design criteria for the PACE-1 chip are shown below as Table 3-1.

Charge Capacity	30×10^6 electrons
Readout Noise	< 500 electrons
Maximum Readout Rate	2 MHz
Operating Temperature	77K
Integration Time	30 ms
Peak Quantum Efficiency	> 70 %
Pixel Operability	95 %
Waveband Of Response (MWIR)	2.0-5.0 μm

Table 3-1. Design Criteria for Hybrid Focal Plane Arrays⁶

The TCM-1000C incorporates an on-board integration capacitor and standard bit bucket-brigade output. This is described in detail in the Rockwell report included in the appendix. The output timing sequence requires three inputs to the chip: master clock (CLK); y-shift register (YSYN), and a framing pulse (FRAME). There are two outputs, however it is possible to read the entire array through only one output by not asserting the FRAME line every 64 clocks, but rather every 128 clocks. The timing diagram for reading the chip is shown in detail in the Rockwell report.

One final feature of note with the TCM-1000C array is the inclusion of an on-chip temperature sensor. This temperature sensor is stated in the report to be uncalibrated,

⁶

For further data about the PACE-1 chip, including dark current data, see the Rockwell report. Dark current is reported to be less than 1 electron per second per pixel at 77K.

however there is a curve showing resistance as a function of temperature (dated 6 March 1979). The output of this temperature sensor is buffered in the RAO-designed preamplifier package and sent to the filter wheel controller⁷ where it meets an ADC. The signal giving the temperature of the chip eventually is passed to the host program running on the PC via a parallel interface card as packed BCD.

3.1.2 Dewar electronics

The TCM-1000C is mounted on a 68-pin carrier which is in turn mounted on a cold finger to keep it at 77 K during operation. However, the 68-pin carrier is integral with an electronics package which provides capacitive decoupling of the bias supplies and rails, and current limiting for the clock inputs. This package has a mini DB-21 connector for which it was necessary to obtain a mate and fabricate a carrier board in order to wire the dewar.

The schematic, taken from the IR Labs documentation⁸ dated October 1989, is reproduced overleaf as Figure 3-2.

3.1.3 Micro Switch

The filter wheel assembly in the dewar is equipped with a micro switch whose actuator slides along the ring gear, as described in Chapter 4. The purpose of this micro

⁷

The filter wheel controller is a microprocessor based assembly designed by Fred Babott. It is not documented in this report.

⁸

Reference 14

switch is to provide indication of when the filter wheel is in position. Normal micro switches are not rated to operate at cryogenic temperatures, however Honeywell⁹ manufactures a line of cryogenic rated micro switches. There is one difficulty with their product, in that the switch is lever actuated and not roller actuated. This requires that the filter wheel turn in one direction only in order to avoid jamming the switch. The output of the switch is not buffered at all in the dewar and is processed in the RAO-designed preamplifier.

3.2 RAO Designed Support Electronics¹⁰

The IR camera is designed to be mounted on the 1.8 metre ARCT, on an instrument cube, one of whose major purposes is to accommodate this camera. However the control room, which is where the controller electronics are located, is several metres distant from the camera. It is therefore necessary to have some support electronics local to the camera to drive the Rockwell TCM-1000C and to buffer the output from this IR array.

The RAO designed support electronics consists of four subassemblies. These are the preamplifier¹¹; a bias voltage generator for the Rockwell IR array; a logic board; and a driver

⁹

Honeywell part number 109HM1

¹⁰

This electronics package was designed in-house by Fred Babott and the author. Layout and assembly was performed by Fred Babott.

¹¹

Actually, a two-channel preamplifier. Detailed description is in Section 3.2.1, next page.

board for the shutter solenoids.

The electronics package is mounted on the side of the IR camera dewar and requires a clean bipolar power supply of ± 12 to 15 volts DC.

The individual subassemblies in this package are discussed in detail below; the schematics and fabrication drawings used in the manufacture of the package are shown in Appendix A¹².

3.2.1 Preamplifier

The preamplifier has two channels to accommodate the possibility that the Rockwell chip may be run in dual-output mode¹³. The channels are identical. Each channel consists of a two-stage amplifier with an AD743 op-amp set up as a voltage follower feeding an LM6181, also set up as a voltage follower, as a line driver. The LM6181 is terminated with a 51 ohm resistor in series with the output, to match the characteristic impedance of the RG-174/U coaxial cable used to connect the preamplifier to the electronics in the control room.

¹²

Photocopy of best copy available.

¹³

The wiring is in place for running the chip in dual-output mode. The software as of this writing is intended to drive the chip in single-output mode. In order to run the chip in dual-output mode, the waveform table in the controller code must be revised and the host software must be modified to de-interlace the outputs and to assemble them into an image.

The advantage to running in dual-output mode is that the chip can be read at twice the rate. The disadvantage is that doing so places an additional software burden on the host. This may be done at a later time, when everything is working.

The preamplifier board also has the circuit for the temperature sensor on it. This circuit uses an Analog Devices AD581 voltage reference to generate a stable voltage for the temperature sensor on the Rockwell chip. The output of the AD581 is buffered by an OP-27 op-amp configured as a non-inverting amplifier. A voltage divider drops the supply voltage to the temperature sensor to 3.4 volts when the TCM-1000C is at 77 K.

The output of the temperature sensor is buffered by another OP-27 op-amp and passed on to the filter wheel controller¹⁴, where it is converted to a digital signal and is transmitted to the host program via a path not discussed in this report.

There is an amplifier enable line (digital) which must be asserted by the IR Labs controller in order to turn on the preamplifiers. This uses half of a 74HC04 package (located on the digital board) and a 2N2222A transistor to turn on the preamplifier.

3.2.2 Bias Generator Board

The bias generator board consists of an LM199 voltage reference, buffered by an Analog Devices OP-27 set up as a non-inverting amplifier. The OP-27 has a 5 K ohm trim pot for fine adjustment. The output of the OP-27 goes to a five-output divider stack, with each divider having a 1N4448 diode used as a snubber and with a 10 K ohm trim pot. Final decoupling for all outputs is with a 10 μ F tantalum capacitor. The output voltages are as shown on the schematic in Appendix A.

¹⁴

The filter wheel controller is a stand-alone rack-mounted assembly designed by Fred Babott, and includes other functions not discussed here.

3.2.3 Logic Board

The logic board is a mezzanine board mounted in the preamplifier assembly, over the preamplifier board. It has on it the level shifters for the logic and also the LM317 and LM7805 regulators for the local power supplies. The 74HC04¹⁵ used to buffer the micro switch is located on this board as well.

The local power supplies are standard configurations as supplied in the sample circuits from National Semiconductor or other vendors.

The level shifters, of which there are four, are TC427 noninverting buffers. Each level shifter has a 1 K ohm current limiting resistor followed by a pair of 1N4456 diodes as snubbers. The purpose of the level shifters is to convert the digital output levels from the IR Labs controller to the requirements for the Rockwell IR array.

3.2.4 Solenoid Driver Board

Since the solenoids driving the camera shutter operate at 77 K, the nominal coil resistance of 5 K ohms¹⁶ (at room temperature) drops to a few tens of ohms. Thus, the operating voltage required is much less than that of the room temperature value of 24 volts. This allows the shutter solenoids to be operated with very low voltages. On the other hand,

¹⁵

Half of the 74HC04 package is used to turn on the preamplifier. The other half is used to buffer the micro switch output.

¹⁶

5 K ohms according to the published datasheet, 136 ohms at room temperature as measured with a DMM, 24.6 ohms as measured at 77 K.

the IR camera is mounted on the back of a telescope that can be pointed in any direction. Thus, not only must the solenoids pull the shutter into the desired position but they must hold it in this position against gravity.

To do this, there are four solenoid drivers. The solenoid drivers are grouped in pairs, with each pair consisting of a throw driver and a holding driver. The drivers are operated by the IR Labs controller which causes the throw driver to assert a high (3 volt) voltage pulse to move the shutter, and then which causes the holding driver to assert a low (0.5 volt) holding voltage to hold the shutter in position against gravity.

Each solenoid driver consists of a TIP120 NPN transistor in a common-emitter configuration with the solenoid coil as the load. There is a 1N4007 diode across the emitter and collector of the transistor to serve as a kickback diode.

3.3 IR Labs Controller¹⁷

The IR Labs controller is a microprocessor based system with two channels of communication to the host computer, and two channels of input from the IR array. It generates the timing signals for the IR array, as well as generating the control signals to gate the preamplifier and to drive the shutter solenoids.

The physical format of the IR Labs controller is that of a 3U VME chassis, however the backplane is not VME as all pins have been redefined to produce a custom bus. The chassis includes power conditioning, however the power supply is an outboard assembly.

¹⁷Schematics are available from IR Labs directly or from their website.

In theory the controller can drive up to 16 2-channel analog input boards, making a total of 32 channels of input from one or more arrays. This capability is useful when driving cameras built by making a mosaic of IR or optical arrays.

There are four types of boards available from IR Labs for this controller. These are: timing board; analog input board; clock generator board; and utility board. The controller to hand uses the first three but is not equipped with a utility board.

3.3.1 Timing Board

The timing board has the microprocessor¹⁸, RAM, ROM, some glue logic, and the communication interface with the host. There can be only one timing board¹⁹ in a controller backplane. The timing board controls the system by communicating with the other boards in the system via the backplane.

The communication interface with the host is designed to mate to a Spectral Instrument Model 1826 PCI card in the host. It has two channels. The first channel is an RS-423 bidirectional serial link running at 9600 baud, and this channel is used to transfer control information between the controller and the host. The second channel is a 16-bit wide

¹⁸

Motorola DSP56002 running at 50 MHz.

¹⁹

The timing board described in this document is a Rev. 3C timing board with wired communication with the host. When dealing with this system it is important to make sure the revision level of the documentation is the same as the revision level for the hardware, and to be aware of various undocumented features of the hardware as received by the RAO from IR Labs (the system they supplied was a design prototype).

parallel channel which, on the controller side feeds directly from the analog input cards under the control of a PLD and on the host side feeds into a DMA device and memory²⁰. Both channels use differential line drivers and receivers. The physical format for the cable between the Spectral Instruments 1826 card and the IR Labs controller is that of a SCSI-3 connector but here, as in the case of the VME bus, all pins have been redefined for the purpose.

There are two types of timing board available from IR Labs. The first type uses the SCSI-3 connector as described above, and is the type used in the camera system described in this report. The second type uses a fibre optic link between the host and the controller. There is only the one interface card available for the host if the SCSI-3 format is used, which requires that the host have a PCI backplane. However, for the fibre optic link, there are several interface cards for several types of hosts (VME, SBUS, PCI), allowing a wide range of machines to serve as the host.

There are some electrical advantages to be derived from the fibre optic link. Since this is an optical link, ground loops, hum, and noise from the copper connection are eliminated. Additionally, the distance restriction between host and controller, placed on the system by the SCSI-3 cable, is removed. Finally, with the optical link, the IR Labs controller may be mounted directly on the telescope near the camera, minimizing hum and pickup.

20

Working with this requires dealing with a VXD device driver (under Windows 98) and a DLL. Details are given in Chapter 7..

3.3.2 Clock Generator Board.

The clock generator board²¹ consists of 24 channels of output. The voltage of each output can be set under program control, and the board is designed so that the outputs may be paired up to form differential sets. Alternatively, the outputs can be referenced to common. There are six blocks of channels on the board, along with some multiplexers and isolation amplifiers intended for troubleshooting. There are also conditioned reference voltage generators, supplying ± 5 volts²².

Each block of channels consists of two Analog Devices DAC8240 serial-load D/A converters. These DACs have four channels of output, and are preset by the timing board during boot or when commanded to do so by the host. After they are set their output voltages do not change.

The outputs from the DAC8240 is then fed to a pair of Siliconix DG613²³ four-channel analog switches, with the channels in four groups of two. From here, groups of two are summed with a summing amplifier/buffer, passed through a Siliconix DG408A analog switch used as an isolation switch, and to the load.

Output from the board is via a DB-27 connector and as stated before there are

²¹

The board described in this report is Revision 4A.

²²

There are jumpers that can be used to set either or both reference voltages to common.

²³

The logic of the DG613 is such that one channel of each pair is closed when the gate is asserted, and the other channel is closed when the gate is deasserted.

multiplexers which are under program control and which can be used for test and troubleshooting.

There are four DIP blocks into which the user can solder protective devices such as diodes if desired. These diodes are referenced to ground through 100 K ohm resistor networks.

3.3.3 Analog Input Board

The analog input board described herein is a design prototype board originally intended to work with the PICNIC²⁴ IR array. The identification on the board describes it as Revision 2C with a layout date of 23 August 1996, however this information may not be correct. The production board uses Datel ADS-937 ADCs whereas the prototype boards²⁵ uses Analogic ADC4325 ADCs.

Each of the two channels on the analog input board is independent, and the two channels are identical.²⁶ Each channel consists of a programmable gain stage, followed by

²⁴

This is another Rockwell IR array, currently (October 2001) in production.

²⁵

The boards at the RAO are the only two boards like this. Of the two, one is believed by this narrator to have a defective channel. The designer of the boards (Dr. Robert Leach, San Diego State University) was unable to recall the reason for making the change from the Analogic to the Datel parts. He did state however that the microprogramming for the production boards is identical to that for the prototype boards, so there should be no problem if it becomes necessary to install production boards in place of the boards now in the possession of the RAO.

²⁶

Except for address on the backplane. Each channel is individually addressable.

a stage which can inject a DC offset, followed by the ADC. The ADC can run under either direct DSP control or under the control of the on-board CPLD, the mode used for doing high-speed transfer to the host.

There are also two DAC8240 D/A converters. These converters, which are serial-load four-channel converters, feed voltage followers. Each group of four channels then passes through a Siliconix DG408A analog switch used as final isolation, and then off-board via a DB-15 connector. The purpose of this circuit is to provide bias voltages for the arrays²⁷.

3.3.4 Utility Board

IR Labs sells a utility card for their system which can be used to generate high (± 30) voltages, support secondary communication channels, and for other purposes. The prototype system this document describes is not equipped with such a card.

3.4 Spectral Instruments Model 1826 PCI Card

The Spectral Instruments Model 1826 card is a PCI card designed to facilitate block transfer of 16-bit words over a differential pair parallel interface. The card has DMA capability, and can write (or, less commonly, read) a specified number of words to or from a specified location in memory.

²⁷

Not used in the application described herein. The bias voltages are made with the RAO designed package.

The card also has a second, low-speed serial channel consisting of an RS-423 interface and an on-board UART. The UART is programmable for a variety of speeds and formats but the standard when dealing with IR Labs equipment is 9600 baud 8N1. It is not necessary to use the on-board UART as there are jumpers and a header that allow an external UART to be connected if desired. If this is done then the card merely provides an RS-232/RS-423 translator.

In a Windows 98 system the card is driven by a VXD driver which is loaded at run time²⁸ and which must be in a directory in the system path. The driver interfaces to a DLL which has calls that an application can access. This is described in Chapter 6 of this document.

3.5 Cabling

The RAO designed electronics package has two gas-tight circular connectors. One connector is for coupling to the electronics inside the dewar, and the other connector is for coupling to the controller. Additionally, two video cables in RG-174/U carry the output of the IR array to the video input card of the IR Labs controller. Pinouts for these connectors are given in Appendix B.

The DB-37 connector on the clock generator board in the IR Labs controller supplies eight digital signals and their returns. The pinout for this DB-37 connector is also given in

²⁸

It unloads automatically when the calling program exits.

Appendix B.

There is a DB-37 connector and a DB-25 connector on the back of the IR Labs controller. The DB-37 connector mates to the cable which goes to the RAO designed electronics package, and the DB-25 connector mates to the cable (not supplied by the author) which goes to the RAO filter wheel controller and the ± 15 volt power supplies. Wiring internal to the controller runs between the DB-37 connector and the DB-25 connector in order to match everything up. Pinouts for these connectors is given in Appendix B.

CHAPTER FOUR
MECHANICAL ASSEMBLY OF DEWAR

Lord, thou has made this world below
The shadow of a dream,
And taught by time I tak' it so-
Exceptin' always steam.

From coupler-flange to spindle-guide
I see thy hand Oh God -
Predestination in the stride
o'yon connecting rod..

Rudyard Kipling

CHAPTER 4.

MECHANICAL ASSEMBLY OF DEWAR

4.0 Introduction

The IR array is housed in a dewar along with a filter wheel, some support electronics, an activated charcoal getter, and a shutter assembly. Since the array is designed to be sensitive into the M band, it must be operated at liquid nitrogen temperatures.

Because the mechanical components of the camera were either obtained from other sources or were developed in the machine shop, engineering drawings of the assembly do not exist. To get around this, photos were taken during assembly. These photos and this narrative are intended to provide the necessary information should reassembly of the camera be required.

4.1 Filter Wheel Assembly.

Refer to Figure 4-1¹. This is a schematic drawing of the filter wheel. There are six positions for filters on the filter wheel. The filter wheel has a ring gear on the outer edge, which engages a spur gear that is turned by a stepper motor. There are notches in the ring gear into which a micro switch² drops when the filter is in position. The closure from the micro switch is used to indicate that the filter is in position, and an outboard controller then

¹

Drawing courtesy Fred Babott.

²

This is a Honeywell cryogenic rated micro switch, part number 109HM1. Because this microswitch uses a sliding lever and not a roller, the filter wheel is constrained to turn in only one direction.

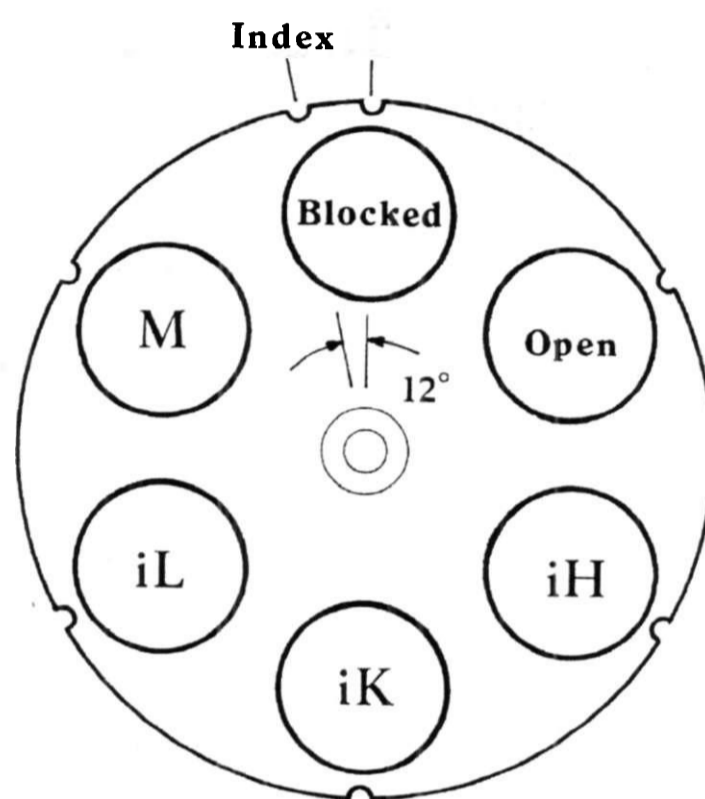


Figure 4-1. Schematic Drawing of Filter Wheel (courtesy F. Babott)

stops the filter wheel precession. The fiducial position is indicated by two notches. There are four filters³ in the filter wheel at this writing, with the remaining two positions being a blank plate and an open position.

The housing for the filter wheel contains two sets of bearings. The first bearing is for the spur gear which turns the filter wheel, and consists of a bushing only. The second

³

The filters are ih, ik, il, and M. Except for the M filter, these filters are narrow-band filters not in the Johnson-Cousins system. Characteristics for these filters are given in Appendix C.

bearing is for the filter wheel itself. This second bearing⁴ is a ball bearing assembly with the ball bearings running in a groove. On the opposite side of the filter wheel from the ball bearings is a teflon thrust washer and a compression spring made from 0.006 inch shim stock. Figure 4-2 shows the front⁵ of the filter wheel assembly.

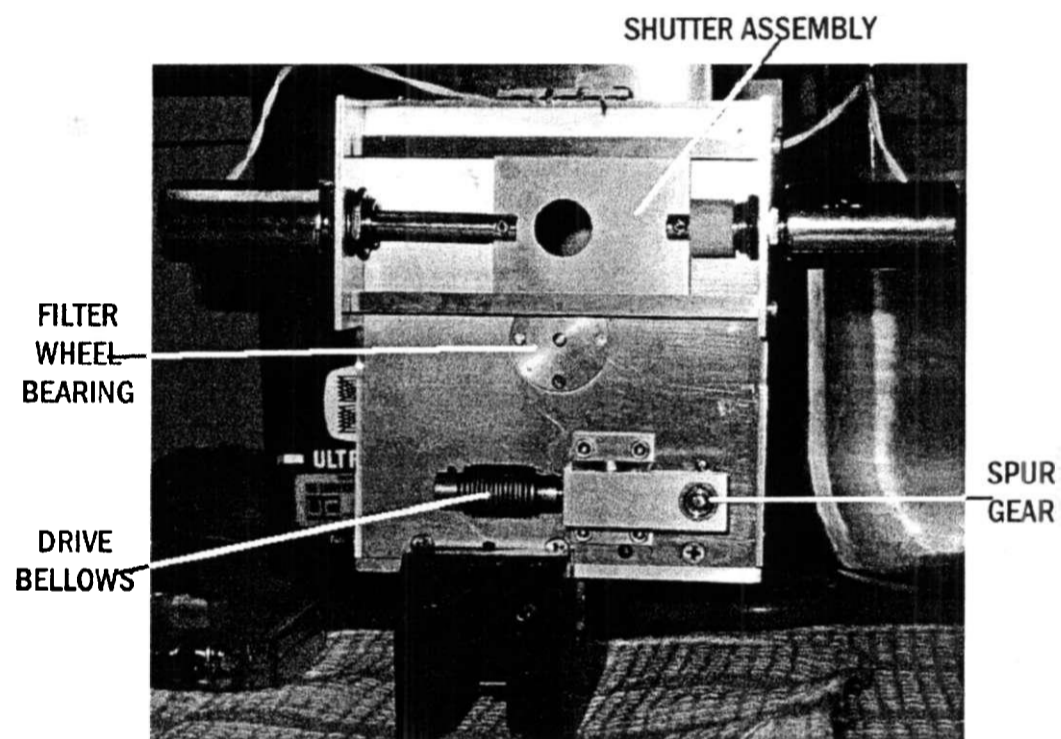


Figure 4-2. Front View of Filter Wheel Assembly

4

If it becomes necessary to disassemble the filter wheel housing you will lose the ball bearings onto the bench you are working over. You should disassemble the housing over a cloth so that the bearings are caught by the cloth. When reassembling the housing, set it up in a Palmer vise so that the race is up and the housing is level. Load the ball bearings into the groove and then mount the filter wheel. The bearings are 1 millimetre in diameter and are available from stock from several vendors.

5

Front is defined as the side facing the light source.

Note the shutter assembly. This assembly consists of an aluminum shutter mounted between two guide rails and driven by two solenoids. There is a teflon spacer mounted on one of the solenoid plungers. This spacer is sized so that when the shutter is opened the hole in the shutter will line up with the IR array.

Note also the spur gear and drive bellows. Inside the filter wheel assembly the gear on the end of the shaft engages the ring gear on the filter wheel, while on the outside of the assembly, two spur gears connect the drive bellows to the stepper motor. The purpose of the drive bellows is to insert some give into the drive train to allow for thermal expansion and misalignment of the drive train. Figure 4-3 shows the reverse side of the filter wheel

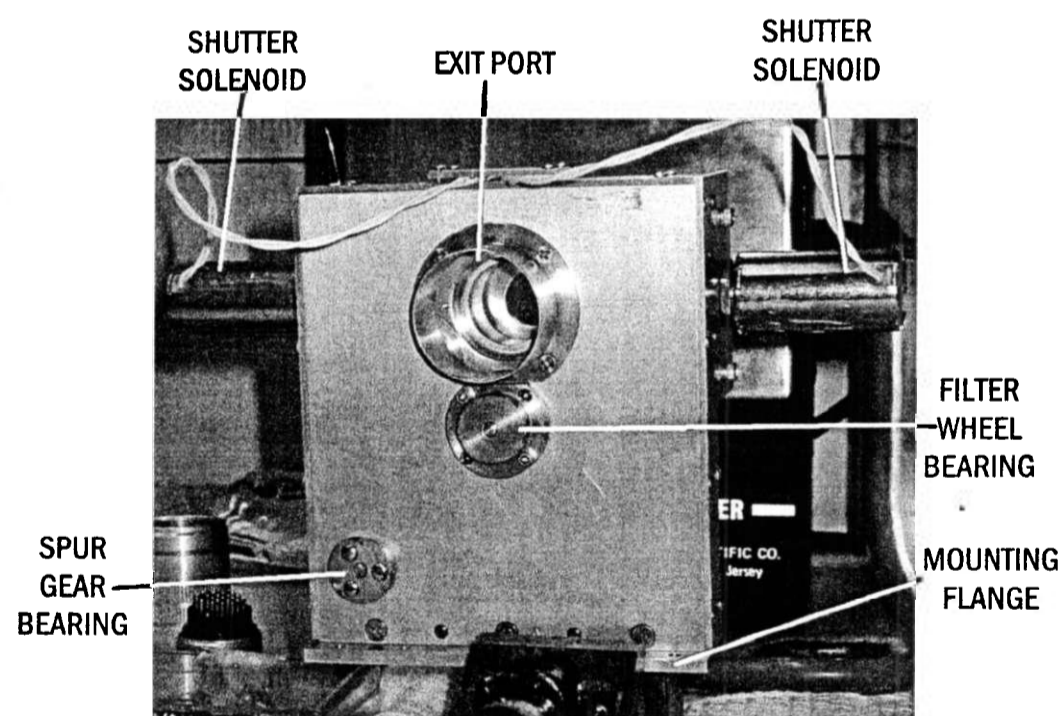


Figure 4-3. Reverse Side of Filter Wheel Assembly

assembly, including the mounting flange.

If it becomes necessary to disassemble the IR camera, the filter wheel assembly is the last assembly which should be removed and the first assembly to be reinstalled. It should be possible to mount the filter wheel assembly to the cold plate while the inner and outer dewar sides are in place. A quarter-inch drive set with extensions and a universal joint is recommended, along with a set of Allen wrenches⁶. The cold plate is drilled and tapped to accept 2-56 bolts. Do not use bolts longer than ½ inch when mounting the filter wheel assembly.

4.2 Cold Plate

The cold plate is a copper plate which serves as the chassis for the IR camera. This plate is mounted on a bellows assembly which in turn is mounted inside another bellows assembly which supports the inner shield for the camera.

It is imperative that the bellows be subjected to as little mechanical stress as possible during assembly of the camera. For this reason it is recommended that during assembly both the cold plate itself and the outer shell ring be immobilized using wooden shims. After the inner shell is mounted to its ring it will be attached to the cold plate via the G-10⁷ standoffs, and the inner shell and the cold plate will form a rigid assembly. Figure 4-4 shows the cold

⁶

Such as a Chapman Mfg. Co. 1316 tool set.

⁷

G-10 is an epoxy/glass- weave material from which printed circuits are manufactured. It is used here for its thermal insulation properties. See Page 56.

plate, with mounting hole locations and wooden shims.

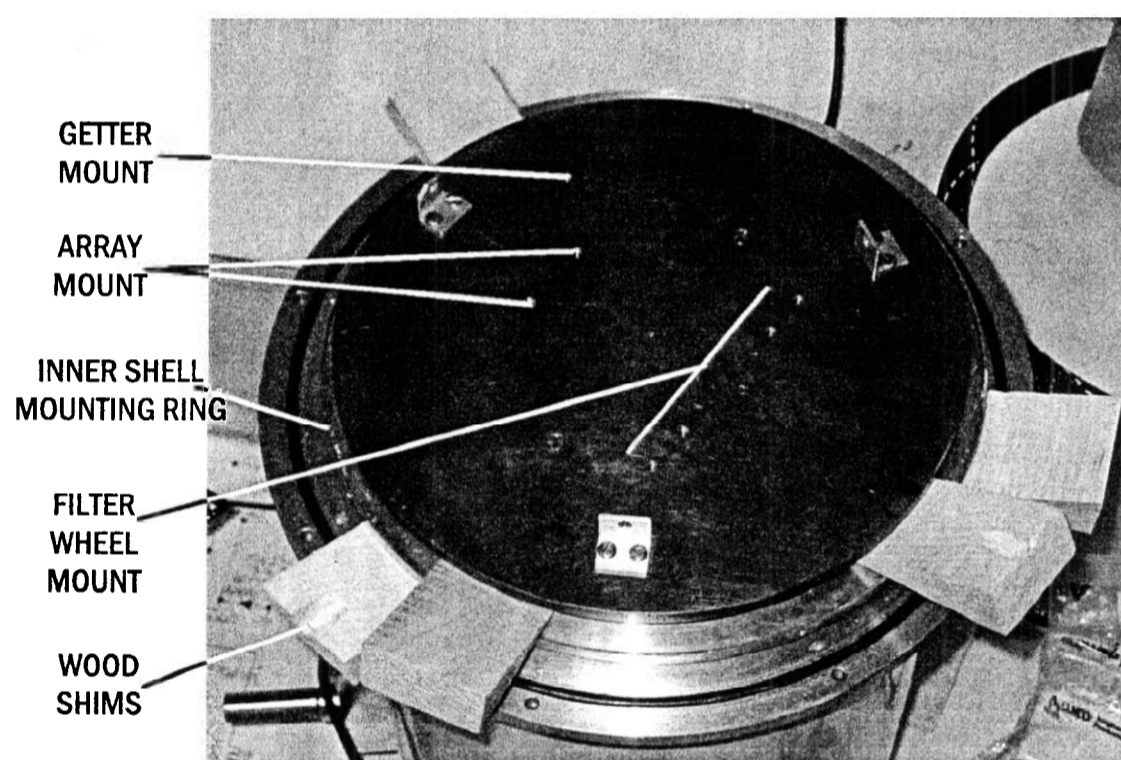


Figure 4-4. Cold Plate

4.3 IR Array Assembly

The IR array assembly consists of a thermal and mechanical link to the cold plate, support electronics for the IR array itself, the chip carrier, a light shield, and assembly hardware. The disassembled view of the IR array assembly is shown in Figure 4-5, overleaf.

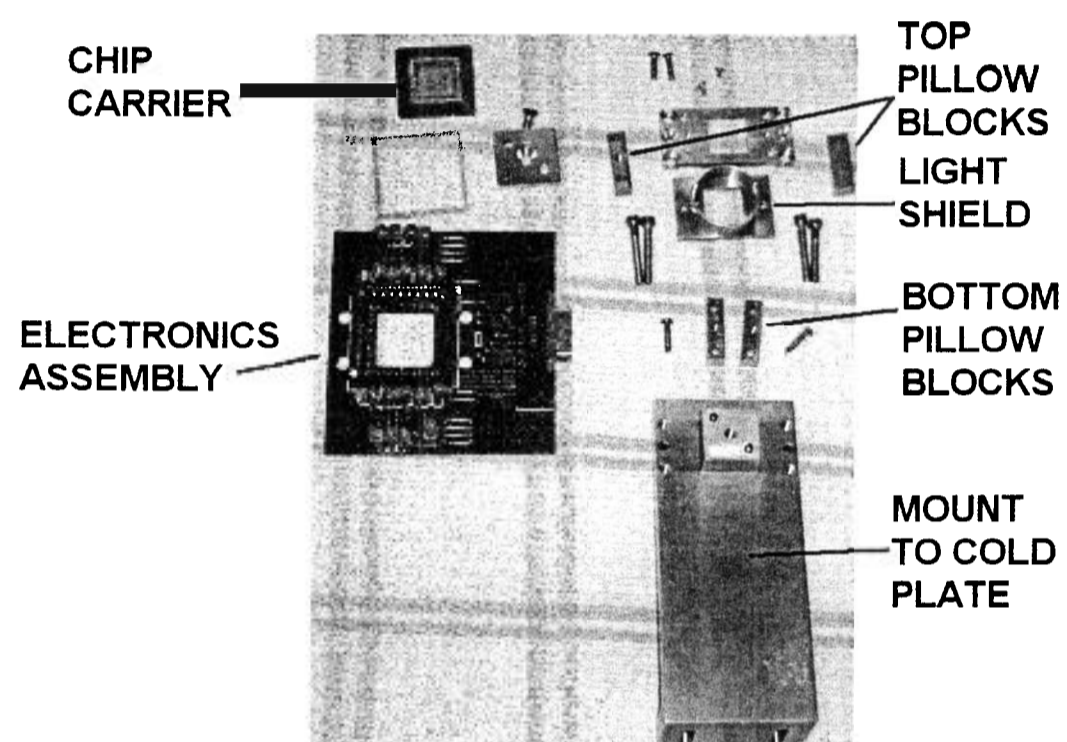


Figure 4-5. Exploded View of IR Array Assembly

Note that the top pillow blocks are not the same size as the bottom pillow blocks, and must not be confused during assembly. The light shield has been notched to clear the shutter solenoid plungers when the assembly is mounted on the cold plate, and care must be taken to ensure that the IR array is parallel to the filter wheel assembly and perpendicular to the light path. The chip carrier will mount only one way in the electronics assembly but the electronics assembly will mount in two ways onto the mount to the cold plate. If the electronics assembly is oriented such that the connector points to the left when mounted to

the mount to the cold plate is as shown in Figure 4-5, a shorter wiring path to the out-of-dewar connector will result. The IR array is not shown in this photograph.

Figure 4-6 below shows the assembled IR array assembly. The scale shown in the photograph is in inches, and the coin shown for size is a Canadian two-dollar coin.

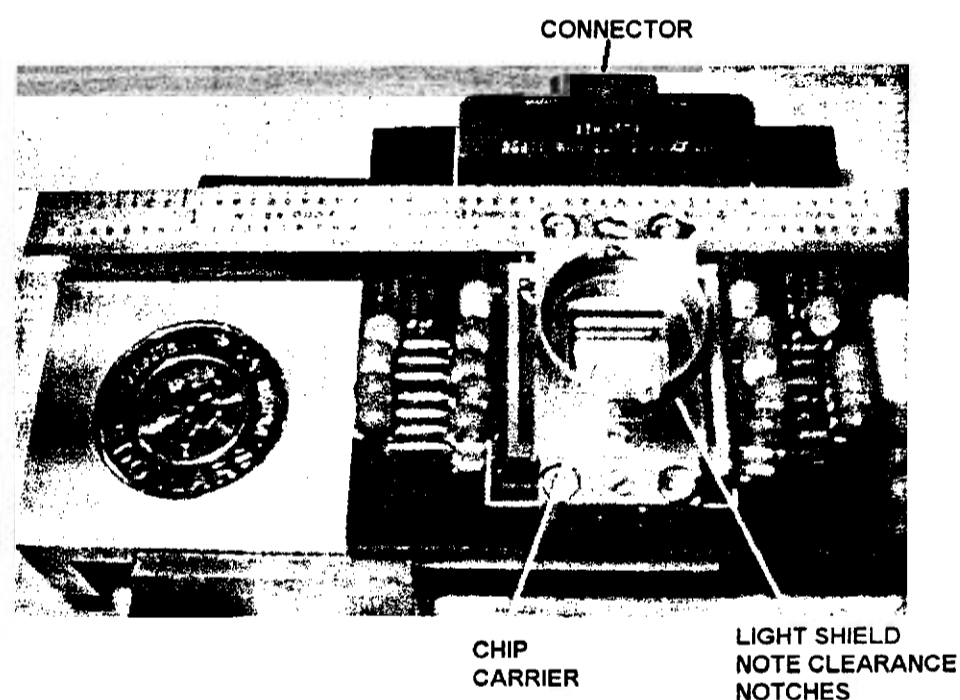


Figure 4-6. Assembled IR Array Assembly

4.4 Inner Assembly

The top assembly of the IR camera consists of mounting the filter wheel assembly onto the cold plate, followed by mounting the inner and outer shells, followed by mounting the drive train for the filter wheel. The IR array assembly may then be mounted, and the

light shield for the IR array may be mounted. The stepper motor and stepper motor controller are external to the dewar and will be discussed in the next section.

Figure 4-7 below shows the filter wheel assembly and the IR array assembly mounted on the cold plate without the inner or outer shells. While this violates the order of assembly described in the previous paragraph, the photograph is instructive in that it shows details of both assemblies and their relationship when mounted on the cold plate.

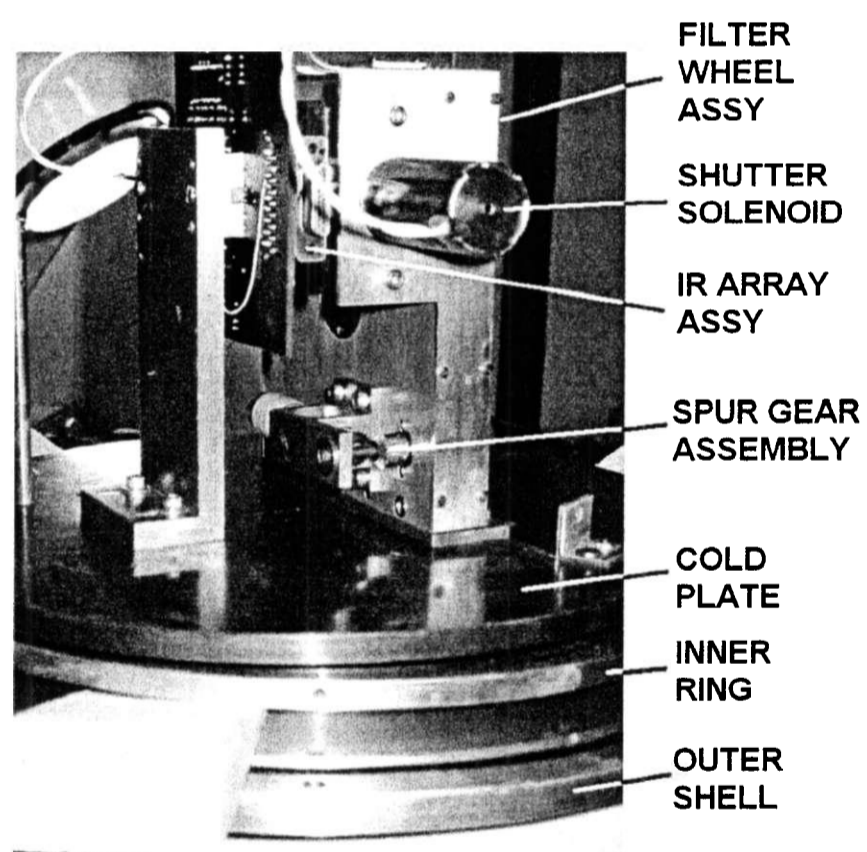


Figure 4-7. Filter Wheel Assembly and IR Array Assembly On Cold Plate

Note that while the filter wheel assembly is mounted to the cold plate with short bolts, the pedestal for the IR array assembly is thicker and requires the longer bolts. These bolts may not be interchanged.

The reason that the order of assembly precludes mounting the inner and outer shells while the IR array assembly is in place is because the setscrews for the filter wheel drive train are not accessible when the IR array assembly is in place. The drive train must be mounted into holes in the inner and outer shells, and then the drive shaft connected to the drive bellows. Only then can the IR array assembly be mounted.

After the filter wheel assembly has been mounted on the cold plate, the inner and outer shields should be mounted. The inner shell is mounted on the inner ring (as shown in Figure 4-7) which is, as described previously, mounted on a bellows not connected to the cold plate.

There are three pieces of 4-40 running thread made from G-10, about 3 centimetres long. These pieces of running thread are used to mount the cold plate rigidly to the inner shell. These pieces of running thread should be mounted with one and only one nut⁸ and one star washer on the outside of the inner shell. The running thread is then mounted in the standoffs on the cold plate with washers and two nuts. Figure 4-8, overleaf, shows a top view of the dewar at this stage of assembly, with filter wheel, inner shell, and running thread in place.

8

If you try to use a jam nut on the running thread on the outside of the inner shell there will not be sufficient clearance to get the outer shell into place.

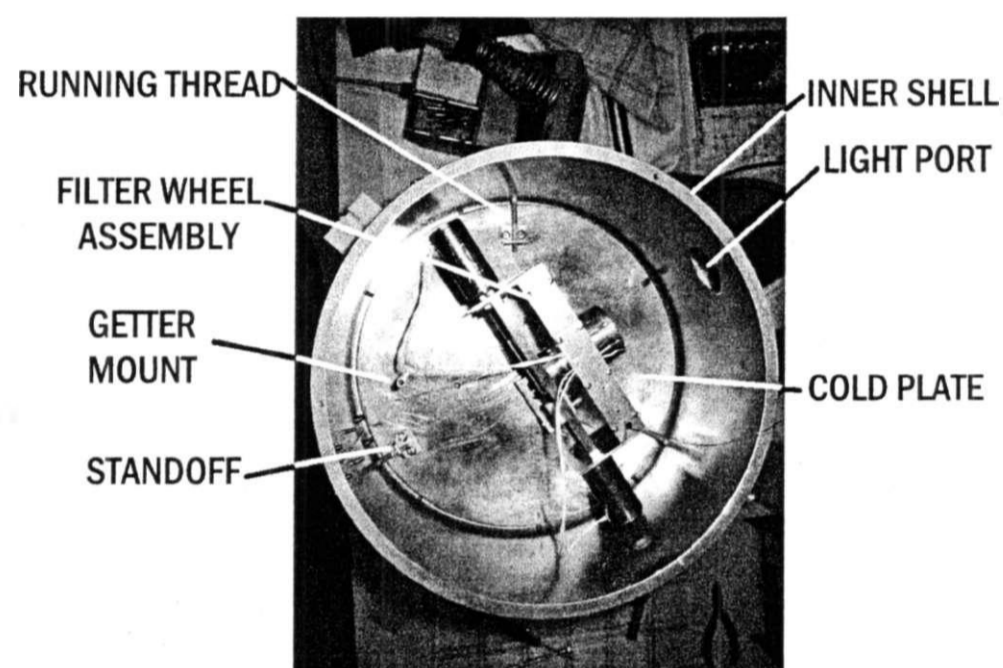


Figure 4-8. Top View Of Dewar With Inner Shell In Place.

The next stage in the assembly is the mounting of the outer shell and the drive train for the filter wheel.

The outer shell seals to the baseplate (not the cold plate) with an O-ring⁹ and vacuum grease. If the plan is to reuse the existing O-ring, it must be carefully removed from its groove in the baseplate. A nonmetallic tool such as a toothpick should be used in order to avoid nicking the O-ring. The ring should then be cleaned and carefully inspected for nicks, cuts, and other defects which would prevent it from making a vacuum-tight seal between the

⁹

This O-ring sizes as a #271 but is actually a #270.

baseplate and the bottom of the outer shell. The groove in the baseplate into which the O-ring seats should then carefully be cleaned of all foreign matter and old vacuum grease¹⁰. The O-ring should then be lubricated with a coating of vacuum grease and re-seated, and the sealing surface should be coated with a thin film of vacuum grease. The outer shell may then be mounted on the baseplate and secured using the bolt tightening pattern shown in Figure 4-9.

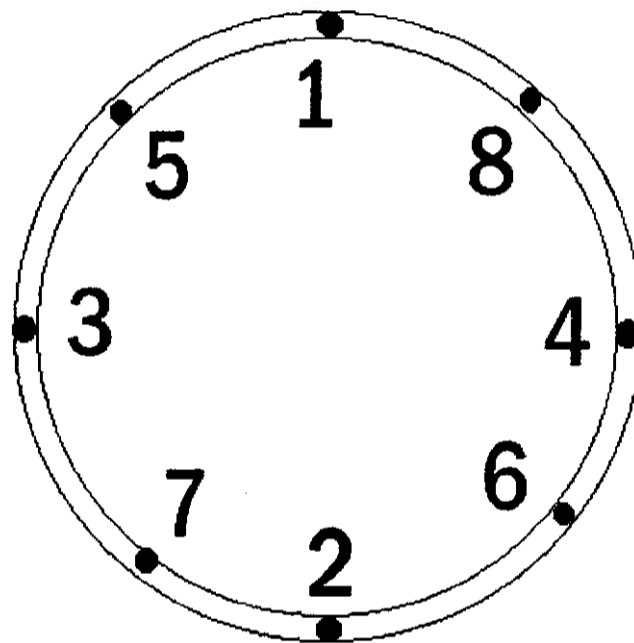


Figure 4-9. Outer Shell Bolt Tightening Sequence

With the inner and outer shells in place, the drive train may now be mounted. The drive train consists of a spindle mounted on a bearing assembly, an O-ring with seal, and a

¹⁰

Ethanol is a suitable solvent.

gland assembly. The housing for the gland assembly seals to the side of the outer shell with the O-ring, and is mounted by four bolts as shown in Figure 4-10. As was done when mounting the outer shell to the baseplate, the O-ring must be removed from its groove, and the groove and all mounting surfaces must then be cleaned. After carefully inspecting the O-ring for nicks, cuts, and other defects, the O-ring should be lightly lubricated with a thin film of vacuum grease and re-seated, and all sealing surfaces should be coated with a thin film of vacuum grease.

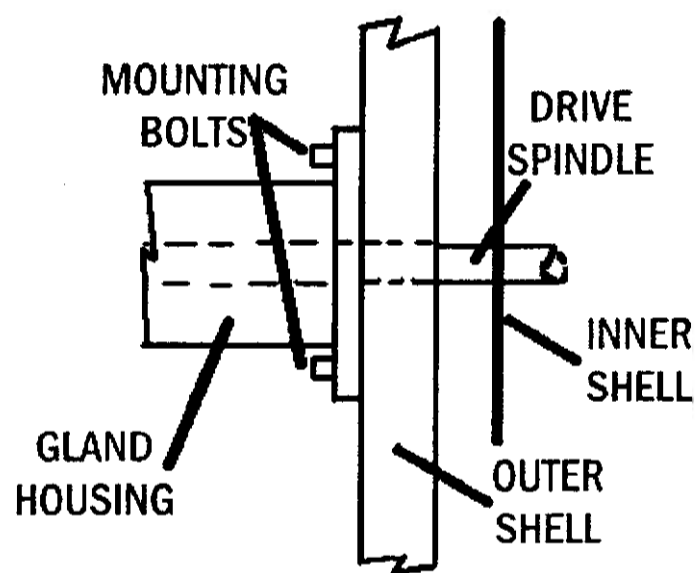


Figure 4-10. Drive Train Mount.

The drive spindle is tipped with a plastic adapter sleeve which mates with the drive bellows on the filter wheel assembly. There are two setscrews in the bellows and one

setscrew in the plastic adapter sleeve. Although the setscrews for the bellows may be tightened without regard to orientation, the setscrew in the plastic adapter sleeve goes into the drive spindle and all holes must be aligned before this setscrew may be inserted. Care must also be taken not to drive the setscrew too far into the plastic adapter sleeve, since if it is driven past the end of the thread, it may not be recoverable, or recoverable only with difficulty. Figure 4-11 shows the dewar with the drive train in place, and shows the location of the setscrews.

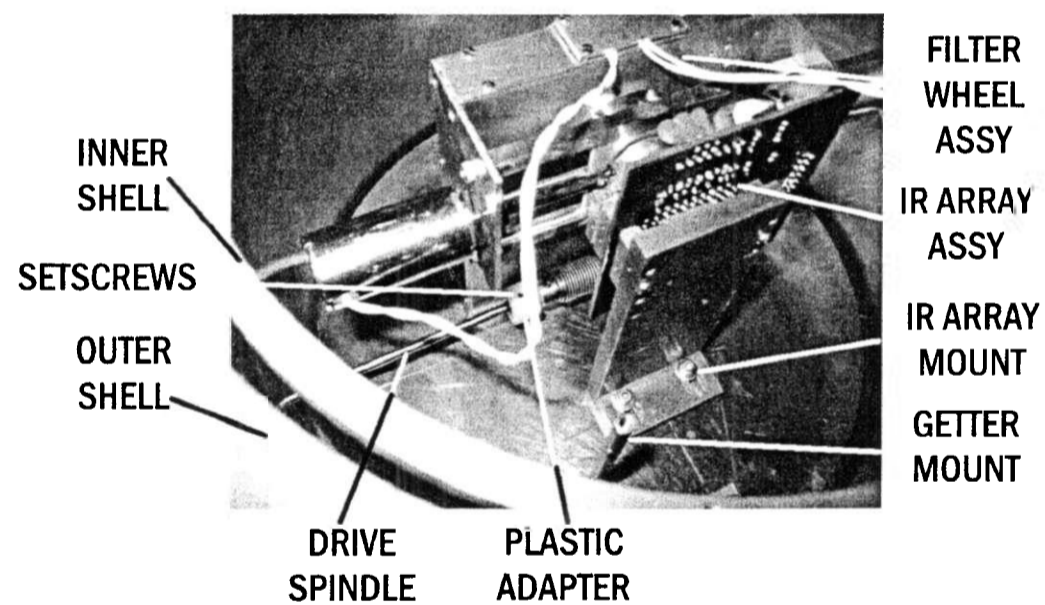


Figure 4-11. Drive Train With Setscrews.

Having mounted the drive train, one can mount the IR array assembly. Care must be taken to ensure that this assembly is mounted square to the filter wheel assembly,

that it is mounted perpendicular to the light path, and that it does not obstruct the operation of the shutter solenoids.

The light pipe and wiring harness may be installed at this time, as shown in Figure 4-12. The getter is not installed until the dewar is ready to be closed up.

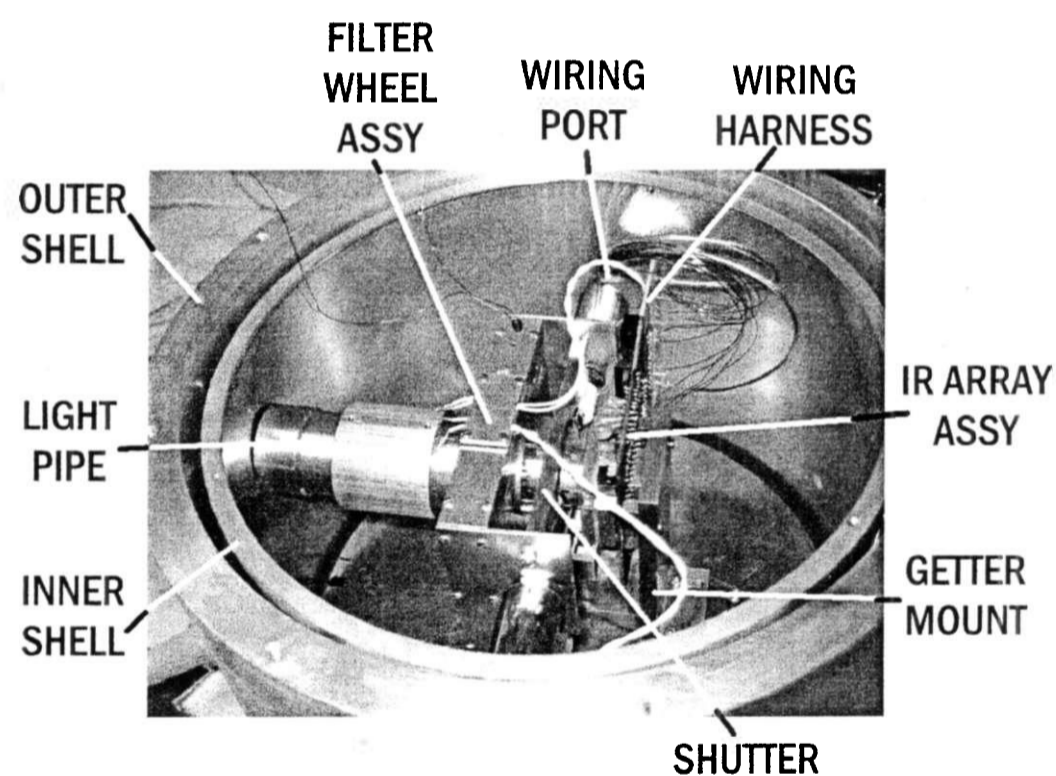


Figure 4-12. Top View of Inner Assembly of IR Camera

4.5 Outer Assembly

The outer assembly consists of the stepper motor assembly for the drive train and associated electronics.

The stepper motor assembly is mounted to the outer shell of the dewar by four brass

standoffs threaded into the shell with 6-32 studs. The motor itself is mounted on the reduction gear housing (as shown in Figure 4-13). The gear train for the stepper motor cannot be assembled until the housing is mounted on its standoffs to the outer shell of the dewar.

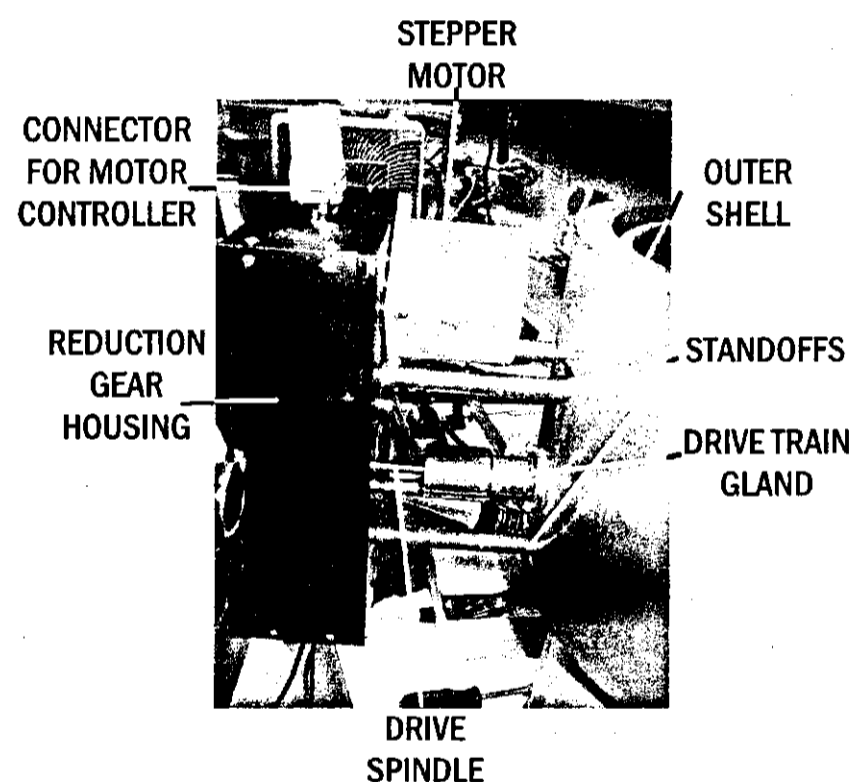


Figure 4-13. Stepper Motor and Reduction Gear Housing

After the reduction gear housing is mounted to the dewar, the reduction gear, drive sprocket, and chain may be mounted inside the housing as shown in Figure 4-14 (overleaf). In this figure, the cover plate for the housing (which has mounted on it a potentiometer for the stepper motor controller) is swung aside and may be seen on the right of the figure.

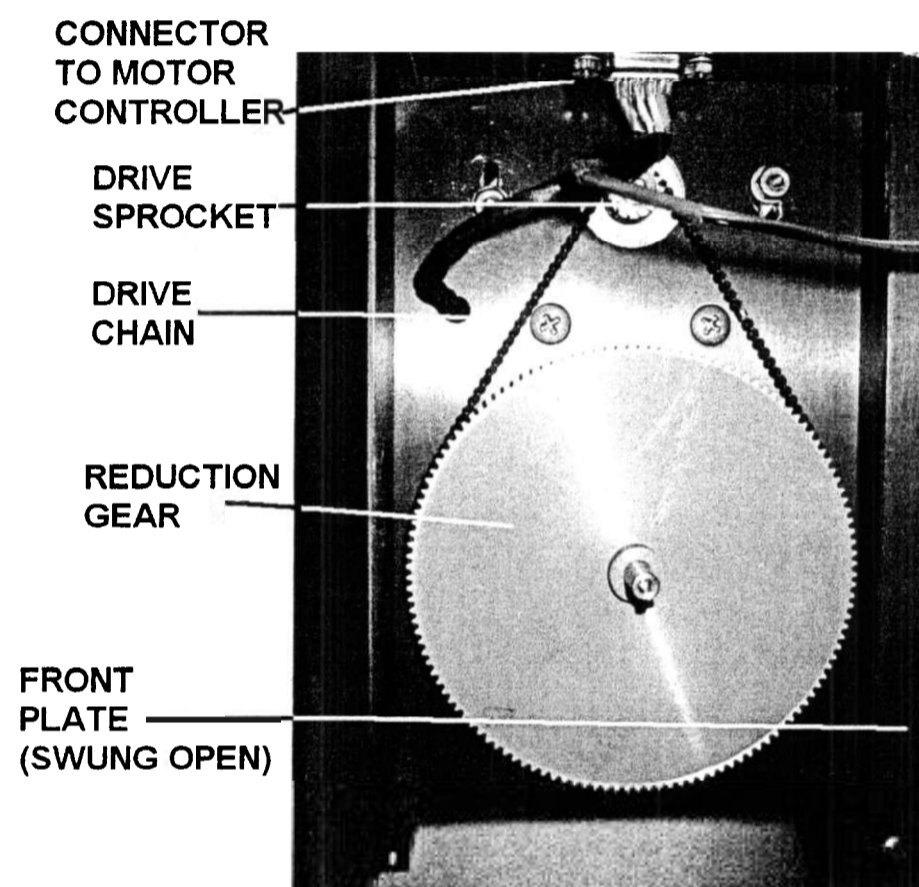


Figure 4-14. Interior View of Assembled Stepper Motor Reduction Gear Housing.

The shaft of the potentiometer on the cover plate for the reduction gear housing engages a shaft on the reduction gear and is held in place with a setscrew. In order to tighten this setscrew it is necessary to reach up from underneath into the reduction gear housing with the appropriate wrench while observing the work with a mirror.

The controller for the stepper motor is mounted below the stepper motor reduction gear housing as shown in Figure 4-15.

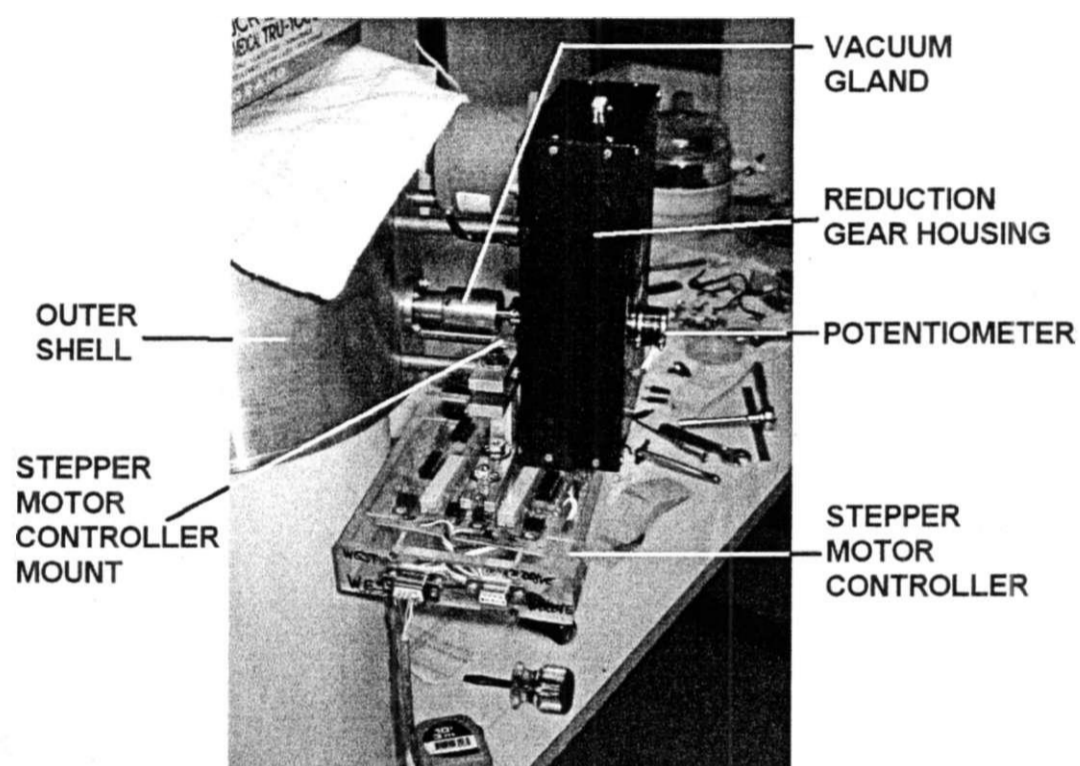


Figure 4-15. View of Completed Filter Wheel Drive With Stepper Motor, Reduction Gear Housing, and Stepper Motor Controller.

4.6 Final Assembly

After the inner and outer assemblies are completed, the getter should be heated in an oven to 150 C for 24 hours. Immediately after it is removed from the oven it should be mounted on the getter mount inside the dewar, and the inner and outer cover plates placed on the dewar. Instructions for preparation of the O-ring for the outer cover, and for preparation of the cover-to-outer-shell interface are the same as for mounting the outer shell

to the baseplate. Tighten the cover bolts using the tightening sequence shown in Figure 4-9.

Invert the dewar so that it is resting on the outer cover plate, connect it to a vacuum pump, pump it down, and check for leaks. If it holds a vacuum then load the inner and outer dewar chambers with liquid nitrogen, let the camera chill down and continue to check for leaks. If there are no leaks then the camera is ready for use. Figure 4-16 shows the completed camera, with electronics package and in its mount.

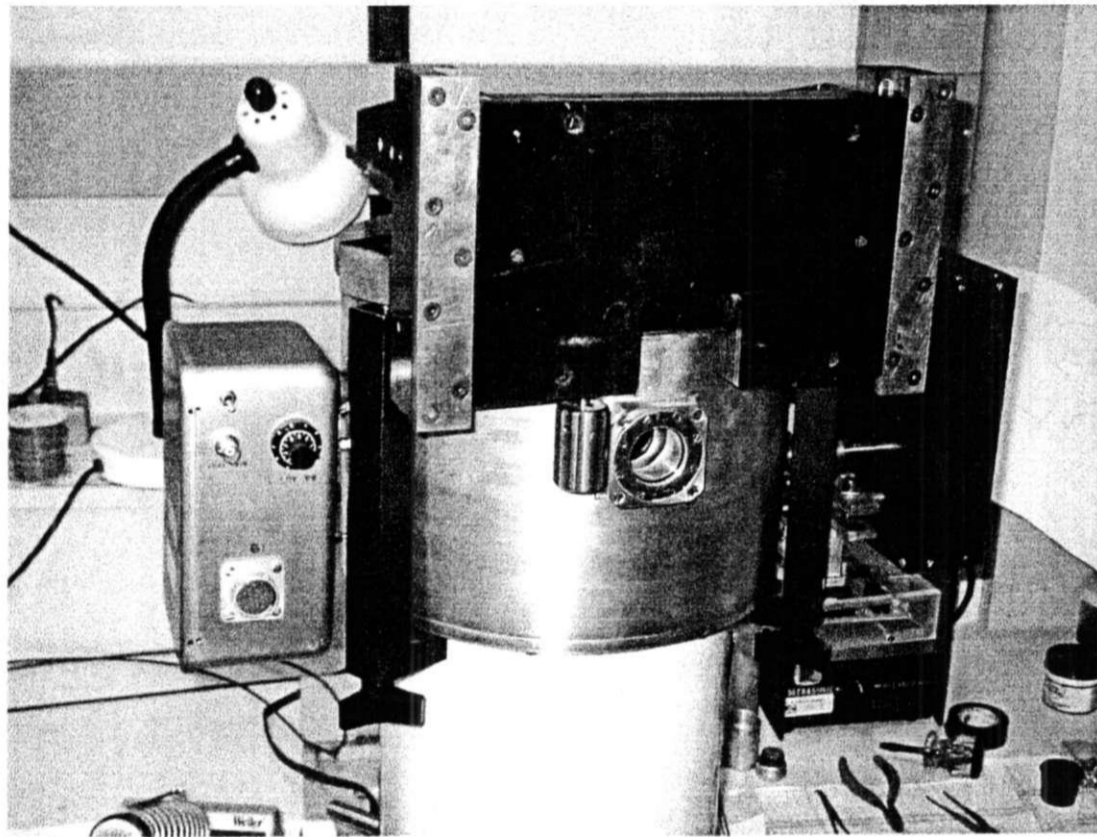


Figure 4-16. Completed Camera In Mount

CHAPTER FIVE

GENERAL CHARACTERISTICS OF A REAL-TIME SYSTEM

When shall we meet again?
In riot, strike or stopping train?
When the hurly-burly's done,
When the insurrection's lost or won.

Barbara Garson, MacBird

CHAPTER 5.

GENERAL CHARACTERISTICS OF A REAL-TIME SYSTEM

5.0 Introduction

Broadly speaking, a real-time system is a software package operating on a platform and controlling or monitoring a process¹. This software package is dedicated to that particular process, and runs in a loop sufficiently tight that it can respond to drifts in the process within the time constant of the process so that the software package can apply corrective guidance to the process and keep the process within an allowed tolerance band for a desired setpoint. To do this, the software package must obtain information about the parameters of the process it is controlling. It must then calculate any required corrective action based upon those parameters and upon information obtained from whatever is controlling the software package, apply this correction action if necessary, and report what it sees upstream to whatever is monitoring the software package.

5.1 Environment

Most real-time systems are embedded systems², although it is not unusual to find a real-time system resident on a general-purpose computer equipped with data acquisition

¹

Wilkes, Ref. 60

²

Embedded Systems Magazine, Ref. 6

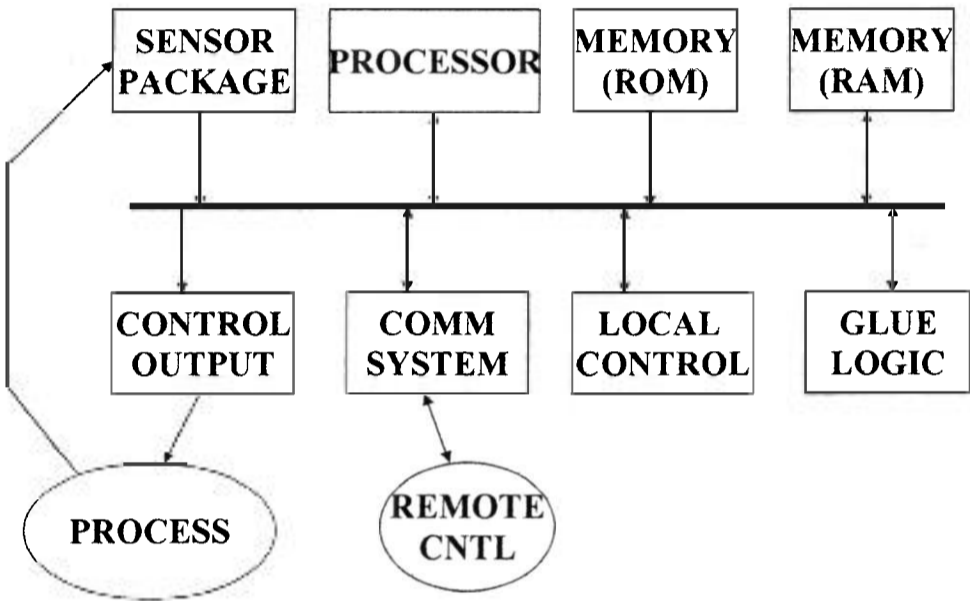


Figure 5-1. Block Diagram of Real-Time System

devices. However, all real-time systems will have a sensor package, local memory, and one or more of the following: communication package; provision for local control; provision for remote control; and the ability to control the process (see Figure 5-1).

5.2 Definition of Real Time Operation

There is no universally accepted definition of real-time operation³, however if a system (software package and platform) has the characteristics listed in this chapter it is generally considered to be a real-time system.

3
Wilkes, Op. Cit.

5.2.1 Time Constant

All processes have a time constant. This time constant is a parameter of the process, not a parameter of the real-time system.

5.2.2 Ability To Stay In Real-Time

The ability to stay in real-time means having the ability to respond properly to the demands of the process under control within the time constant of the process. If this does not happen the system will be deemed to have lost real-time.

5.3 Mission of the Real-Time System

Any real-time system has a mission⁴. This mission requires the ability to perform the tasks listed below.

5.3.1 Control of the Process

This is interpreted as specified above, to keep the process associated with the system within a specified error-band of a specified set point e.g., as discussed by Ott⁵.

⁴

Higgins, Ref. 10.

⁵

Ott, Ref. 51. goes into great detail on statistical methods for process control. A technique he discusses which is particularly useful concerns control charts and how to determine if a process is within tolerance limits.

5.3.2 Control of the Instrument

A real-time system can be considered to be an instrument. Within this context, there is housekeeping to be done to keep everything running. This housekeeping is considered to be operational overhead, is a fixed demand on the resources of the system, and must be accounted for when budgeting the resources of the system.

5.3.3 Take Data.

In order to control a process, the system must be able to acquire data concerning what the process is doing at any given time.

5.3.4 Interpret Commands

The system must be able to receive and interpret commands from whatever is controlling the system, and to implement them. This will require additional machinery within the system in the form of message crackers, task dispatchers, communication subsystems, or other features as appropriate to the system.

5.3.5 Respond to Real-World Events

Not every situation that arises with a process can wait until the system gets around, in the fullness of time, to monitoring it. Transients and other events can occur which require immediate attention and prompt corrective action of some type. This requires that the

system be able to deal with emergencies and anomalies so that the process stays under control.

5.3.6 Respond to the Communication Subsystem

Usually in a real-time system the communication package is a subsystem. This subsystem is frequently interrupt driven and such interrupts must be dealt with promptly. The interrupts that the communication subsystem generates are not emergencies, but are service requests. However the resource requirements of the communication subsystem must be accounted for when budgeting the resources of the system.

5.3.7 Respond to NMI Emergencies

An NMI (Non Maskable Interrupt) is an emergency interrupt of the highest order. This interrupt is reserved for catastrophic failure of some nature, and is almost invariably a shut-down signal. When this interrupt occurs, the system must attempt to put the process into a safe mode, put itself into a recoverable state, communicate to its controller that it is going down, and wait for the end.

5.4 Internal Structure of the Real-Time System

In order to accomplish its mission, the system will require the following major functional blocks⁶. Implicit in this discussion but not directly addressed are such functional

⁶Reference 56

entities such as queues (circular and FIFO) and stacks (LIFO) or the fact that there is a high degree of communication between these functional blocks.

5.4.1 Interface To Hardware (BIOS)

A BIOS (Basic Input/Output Subsystem) such as specified by Microsoft⁷ is an interface between the hardware and the real-time system itself. The BIOS can take the form of code resident in non-volatile memory and included as a component of the platform, or as a fixed component of the real-time system. A typical BIOS (e.g., the System/360 DOS/VSE BIOS⁸) has two layers. The first layer is known as the Physical I/O Control Subsystem (PIOCS). The second layer is known as the Logical I/O Control Subsystem (LIOCS). Both the PIOCS and the LIOCS are open structures in the sense that the real-time system can access any function in either layer as required. A block diagram of the BIOS is shown overleaf as Figure 5-2.

5.4.1.1 Physical I/O Control Subsystem.

The Physical I/O Control subsystem is that portion of the BIOS that interfaces with the hardware and that deals with the specific characteristics of each device. This is the layer in which all the low-level data manipulation occurs, and it provides a low-level standardized interface to the rest of the system. It does not however provide any advanced

⁷Reference 30

⁸Reference 16

functions. If the hardware configuration changes, it is this layer which must be changed in order to accommodate the changes in the platform.

BASIC INPUT/OUTPUT SUBSYSTEM

- **PHYSICAL I/O CONTROL SUBSYSTEM (PIOCS)**
- **LOGICAL I/O CONTROL SUBSYSTEM (LIOCS)**

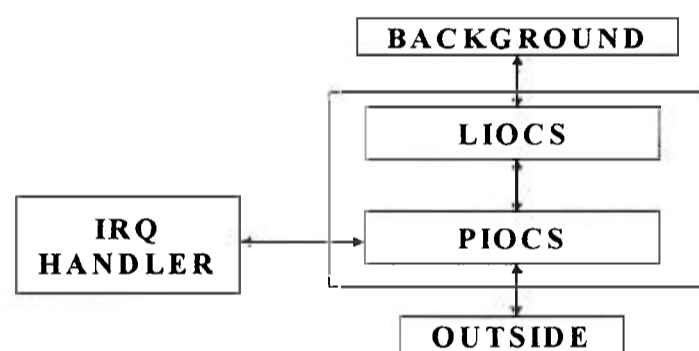


Figure 5-2. Basic Input/Output Subsystem Block Diagram

5.4.1.2 Logical I/O Control Subsystem

The Logical I/O Control Subsystem is that portion of the BIOS which provides advanced functions to the real-time system (or other application). This layer provides a high-level interface between the standardized low-level interface provided by the LIOCS and the real-time system. Often the LIOCS provides extensive capabilities not possible to a simple device driver.

5.4.2 Interface to Communication Subsystem

The communication subsystem (such as the system used in the Texas Instruments TMS320C6000⁹) can be anything from a simple parallel interface to a 100BaseT ethernet connection. Whatever it is, this interface (which works in conjunction with the BIOS) deals with the physical layer and presents message packets to the real-time system. It also takes message packets generated by the real-time system and passes them on to the communication subsystem for transmission upstream. In the following discussion we shall not discuss advanced features such as multiple buffers and message queues, except to note that if the message traffic is sufficiently high, such techniques are available and useful. A block diagram of a typical double-buffered communication subsystem is shown below as Figure 5-3.

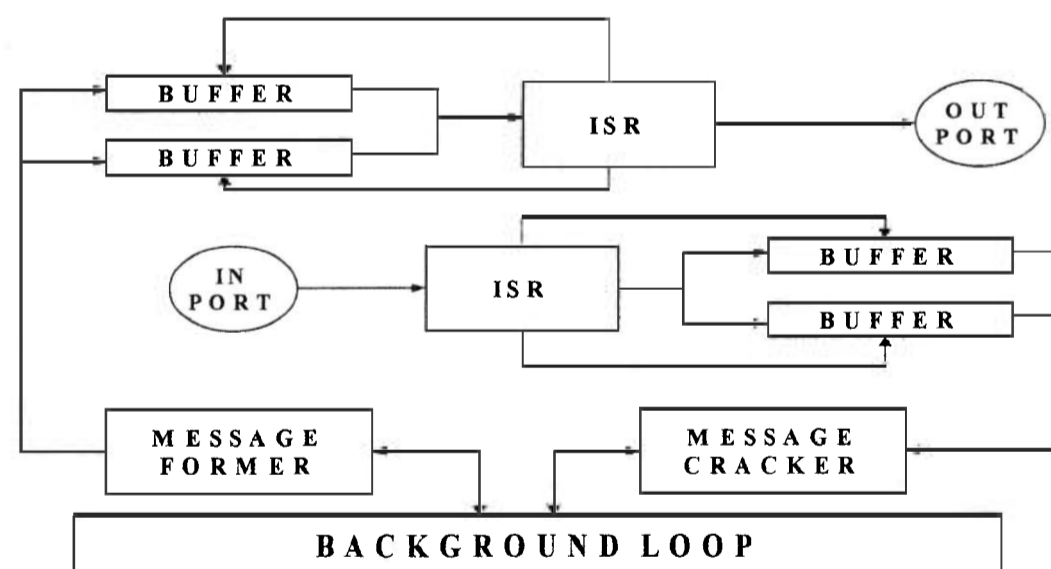


Figure 5-3. Typical Double-Buffered Communication Subsystem

⁹Reference 58.

5.4.2.1 Tick-Tock.

The Tick-tock¹⁰ is the name given to the interrupt handler routine associated with the communication subsystem. It is given this name because transmission or receiving of data is controlled by interrupt.

5.4.2.1.1 Transmission

In the case of transmission, the device buffer is initially empty, which would normally result in a device-ready interrupt, except that this interrupt is kept disabled by the communication subsystem. When a message is ready for transmission, the message is loaded into a buffer and the interrupt is enabled. This results in a Device-Ready interrupt, and the interrupt service routine loads as much of the message into the device buffer as the device is designed to handle (byte or packet, depending upon the nature of the device). The interrupt service routine then leaves the interrupt enabled and executes an RTI command.

When the device buffer empties, another Device-Ready interrupt will occur and another portion of the message will be loaded into the device buffer by the interrupt service routine, which will execute an RTI command, leaving the interrupt enabled.

This sequence will continue until the entire message has been transmitted. When the device buffer empties and the message buffer is empty, the interrupt service routine will disable the Device-Ready interrupt and execute an RTI command.

Note that it is the responsibility of the routine generating the message to ensure that

¹⁰Reference 18.

the interrupt is initially enabled. After that, the communication subsystem will take care of everything else.

5.4.2.1.2 Receiving

In the case of receiving a message, the receive interrupt is always enabled. The device incoming buffer will fill as data (byte or packet) is picked up from the outside. When this buffer is full, the receive interrupt will be triggered and the interrupt service routine will unload the device buffer to a message buffer. The interrupt service routine will then inspect the message buffer to determine if a complete message is present, if any transmission errors have occurred, and if the message is internally consistent (i.e., if checksums and CRCs are valid). If a complete message is present and it passes all validity checks then a message-available flag is set and the buffer is released to the real-time system (usually the message cracker) for further processing.

5.4.3 Message Cracker

The message cracker¹¹ is that portion of the real-time system which interprets incoming messages and determines what they mean. There are two major variants to the type of action the message cracker can take upon decoding a message. A C-language example of a typical message cracker is shown overleaf as Figure 5-4.

¹¹Reference 52.

5.4.3.1 Message Cracker Takes Direct Action

In this case, the message cracker, after decoding the incoming message, directly executes the action requested by the message. This requires that the message cracker incorporate a task dispatcher to ensure that a routine which will perform the requested action is invoked.

```

while(bfin_cracker == TRUE)
{
    switch(inmessage)
    {
        case turn_on_pump:
        {
            bfpump_on = TRUE;
            bfmsg_avail = TRUE;
            break;
        }
        case turn_off_pump:
        {
            bfpump_on = FALSE;
            bfmsg_avail = TRUE;
            break;
        }
    }
}
/* end of switch */
/* end of message cracker */

```

Figure 5-4. Typical Message Cracker

While this approach is conceptually the simplest and is frequently implemented, it has the disadvantages of not allowing for priority based execution of requested actions and

of the possibility that the system can lose real-time if incoming message traffic is sufficiently high that the real-time system cannot keep up.

5.4.3.2 Message Cracker Sets Event Queue

In this case, the message cracker does not incorporate a task dispatcher but loads an event queue with a request flag. The dispatcher is then part of the background loop. The salient feature of this approach is that a prioritizer can be included in the message cracker which can take into account any tasks already in the event queue; the state of the real-time system, and any other pertinent information, and which can shuffle the sequencing of the events in the event queue so as to best conform with the requirements of the mission. The background loop will then pull tasks off the event queue and run them through the dispatcher, which will then invoke the routine to perform the requested action.

Note that while this approach is more flexible than that of the previous section it carries with it a certain amount of overhead which must be accounted for when budgeting system resources. Note also that there is a certain execution latency in this approach which, while deterministic in the sense that the latency is a function of the time required to crack the message, load the event queue, take the event off the queue and execute it, is not deterministic in the sense that, because of the presence of the prioritizer, the latency is a function of whatever else may be happening to the system. For this reason, one thing the prioritizer must monitor (if a prioritizer is present) is the age of the event - the older an event is, the higher its priority must become.

5.4.4 Interrupt Handlers

There are certain interrupts which are required as part of the structure of the real-time system. These interrupts are the communication subsystem interrupts (either one or two, depending on the structure of the communication subsystem); the clock/timer interrupt; and the non-maskable interrupt. There may be other interrupts present as required by the mission of the system.

Whenever the system is dealing with an interrupt, it is not in real-time. All resources of the system are devoted to dealing with the interrupt, and all other functions are blocked. Usually, this blockage extends to the servicing of other interrupts as well (most processors, for example, enter the interrupt service routine with all interrupts disabled). For these reasons, it is of the essence that interrupt handlers be kept as short as possible, that they do only what is required to handle the event which caused the interrupt, and that they return as quickly as possible to the normal flow of execution.

Normally, an interrupt service routine will clear the condition which caused the interrupt, set an event flag in the appropriate queue for further processing, re-enable the interrupt, and exit. The event flag is now what is known as a “soft interrupt”, and is handled by the background loop in a less immediate manner than in an interrupt service routine. A block diagram of a typical interrupt handler is shown overleaf as Figure 4-6.

5.4.4.1 Clock/Timer Interrupt Service Routine

There is one major exception to the above, and that exception is the structure of the clock/timer interrupt service routine. The clock/timer interrupt service routine is frequently used for more than simply updating the system clock. Certain control algorithms (e.g., PID loops) require execution at well defined intervals and timing jitter is deleterious to the operation of these algorithms. More precisely, it is necessary to obtain data as to the state of the process at regular intervals, and it is necessary to output control signals also at regular intervals. These functions are therefore linked to the clock/timer interrupt such that

INTERRUPT HANDLER

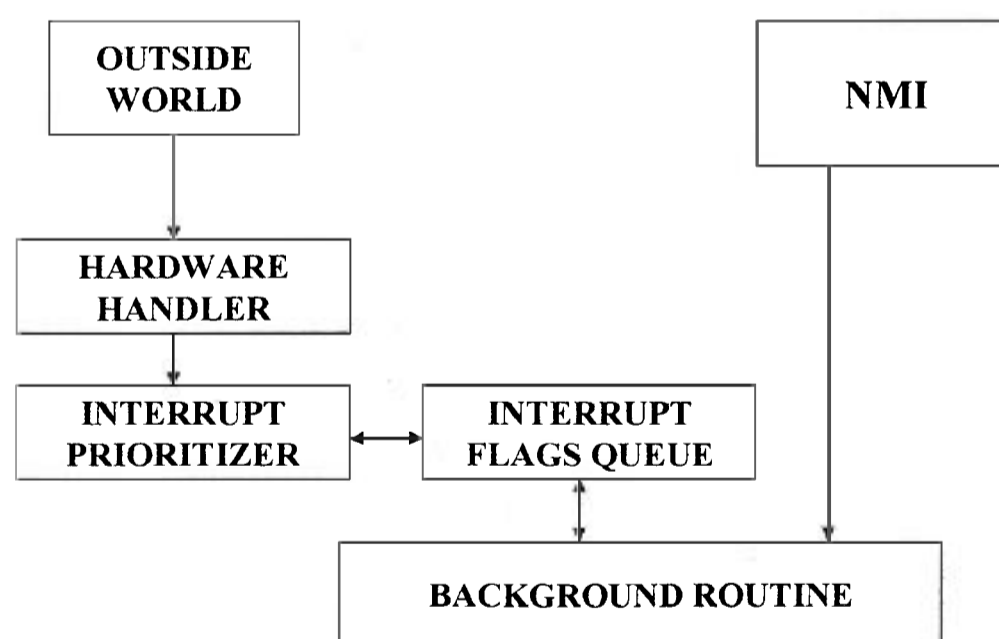


Figure 5-5. Typical Interrupt Handler

they occur on the tick¹² and with as little jitter as practicable. The calculations that make up the control algorithm can be performed under the control of the background loop with the sole caveat that they be done and the control output ready by the time of the next clock interrupt. Failure to have the control output ready by the next tick will result in losing real-time.

Another use for the clock/timer interrupt is to queue events in a virtual machine. This topic will be discussed later in this report.

5.4.5 Process Control Routine

As remarked in Section 5.4, data is acquired concerning the state of the controlled process on the tick. This data is then processed by the control algorithm and is output to the process on the next tick. A frequently encountered control algorithm is the PID (Proportional-Integral-Differential) control loop such as that given by Franklin and Powell¹³. In this loop (see Equation 5-1 of Figure 5-6), the proportional term is a measure of what the process is doing at the time t , with a control multiplier of k_p . The integral term, with multiplier k_i , contains the memory of the PID loop, and allows a momentum in the phase

¹²

That is, at a regular interval as defined by the frequency of operation of the timer interrupt. The standard PC timer frequency is 18.3 Hz, but the PC has an Intel 8253 timer with three channels. One channel normally is used for the memory refresh and one channel is used for the timer. This leaves a spare channel that can be used for user applications. The Microsoft Visual C++ compiler has intrinsic timer routines in its libraries.

¹³Reference 3.

space of the control loop to exist, without which the algorithm cannot meet the target value. The differential term, with multiplier k_3 , is there to deal with transients and usually serves as a transient damping term. The multipliers k_1 , k_2 , and k_3 are usually under operator¹⁴ control and are used to tune the operation of the control loop.

The Laplace Transform form of the PID loop is shown as Equation 5-2 of Figure 5-6. In this form we see that the PID algorithm takes the form of a quadratic equation, and indeed in Equation 5-3 this equation can be factored into the form shown, where a_1 and a_2 represent time constants. If k_3 is zero then the remaining time constant is k_2/k_1 , and this is the critical time constant, in the sense that this time constant must be less than the intrinsic time constant of the process being controlled in order to prevent runaway.

Other control algorithms are possible: the above is simply an example.

5.4.6 Background Loop

The background loop is where the real-time system is when it is not actually handling an interrupt. The background loop interrogates the event queues and, if so equipped, runs a task dispatcher to execute the tasks requested by the items in the event queue. Additionally, it runs the control algorithm and the message cracker. It may also have additional functions as required by the mission, such as display controller, operator

¹⁴

In a manually tuned PID loop. Once the tuning parameters are set they usually do not change. However, PID controllers which have autotune capability are becoming increasingly common.

interface, or other tasks.

PID CONTROL EQUATION

$$f(t) = k_1 i(t) + k_2 \int_{t_0}^t i(t) dt + k_3 \frac{d}{dt} i(t) \quad (\text{Eq. 5-1})$$

$$\frac{V_{out}}{X_{in}} = \frac{1}{k_3 p^2 + k_1 p + k_2} \quad (\text{Eq. 5-2})$$

$$T(p) = \frac{1}{(p + a_1)(p + a_2)} \quad (\text{Eq. 5-3})$$

Figure 5-6. PID Control Loop Equations¹⁵

5.4.7 Virtual Machine

The virtual machine is a concept wherein the request flag in the event queue does not result in a simple operation, but rather in the execution of a series of events, under timer/clock interrupt control. In the form implemented in the DRM-210¹⁶, this system used strings of pseudo-ops which were inserted into a special event queue one at a time by the

¹⁵Reference 50.

¹⁶Reference 18.

timer/clock interrupt service routine. The background loop would unload these events from the event queue and perform the simple task specified by the pseudo-op. This system works well and allows execution of quite complex strings of events with the transmission of a single command from the control source.

Variants of this system can be found in software associated with real-time digital signal processing, and with other control applications.

The linkage sequence for the DRM-210 virtual machine is shown as Figure 5-7, overleaf.

5.5 Commercially Available Real-Time Operating Systems.

Commercially produced real-time operating systems are available for almost all DSPs and microprocessors on the market. These operating systems are true operating systems in that they provide all the services listed above. They are intended for applications where the environment is such that the system can afford the performance overhead in using a canned system, and where the application is sufficiently complex that it is not cost-effective to develop a custom system. They also generally provide for real-time task reload, in that one more control program can be loaded and executed, or control programs can be downloaded from the control source or otherwise specified at or during runtime. There is the advantage that these systems generally allow the developer to write code in a high level language such as C, C++ or ADA. This has the further advantage from a cost perspective in that large portions of the system can be tested in environments other than in the target

```

*          THE LINKAGE SEQUENCE          *
*
*      HASHTABLE:
*
*      HASHTABL:
*
*          [ ] HASHCODE          |
*      (incoming) ———> [ ] [ ] [ ] MNEMONIC      | x 128
*
*          | -- [ ] RESULT CODE      _|
*
*          | ————— |
*
*      VIRTUAL MACHINE: |
*
*          | ————— |
*
*      VIRTABL: \ /
*
*      | ————— [ ] [ ] VIRTUAL PSEUDO-OP STRING ADDRESS
*
*      | (PSEUDO-OP STRING ADDRESS):
*
*      | ————— -> [ ] [ ] [ ] [ ] [ ] [ ] PSEUDO-OPS
*
*          | ^          VIRTUAL MACHINE PROGRAM COUNTER
*
*      | ————— |
*
*      VIRTREAL:
*
*          [ ] [ ]
*
*          [ ] [ ]
*
*      | ————— -> [ ] [ ] —————> REAL MACHINE ROUTINE WHICH ACTUALLY
*
*                                     DOES THE WORK
*
*      +-----+
*
*      ENDIT

```

Figure 5-7. Linkage Sequence for Virtual Machine in RTOS in NRC DRM-210¹⁷

environment.

Examples of this class of operating system include SpoX¹⁸, VRTX¹⁹, and RTOS²⁰.

5.6 Roll-Your-Own Operating Systems

Strictly speaking, a roll-your-own system should be considered only when one or more of the following conditions obtain:

- (1) Expected low production volume causes the develop/purchase tradeoff to be such that it is more cost-effective to do in-house development.
- (2) The task is so simple that a purchased system cannot be justified.
- (3) The characteristics of the embedded microprocessor/DSP/microcontroller are such that it is not feasible to use a commercial product.
- (4) Performance constraints are such that it is not feasible to use a commercial product.

In these cases, the developer will be required to develop in assembler language (there is no getting around this), and possibly C. Assemblers are available for all devices, and in most if not all cases, a C compiler is also available.

¹⁸Spectron Microsystems, Reference 56

¹⁹Mentor Graphics

²⁰QNX Software Systems, Ltd.

Never listen to or take the advice of anyone under 30 as they
may have listened to or taken the advice of someone over 30.

Anonymous graffiti, corner of Haight Street and Asbury Avenue,
San Francisco, 1969

CHAPTER 6.

IR LABS CONTROLLER PROGRAM

6.0 Introduction

The IR Labs controller is built around a Motorola 56002 DSP operating at a 50 MHz clock rate. This processor employs a modified Harvard architecture in that program memory and data memory are separate, however there is not one but two data spaces. There is on-board workspace for program memory and both data spaces and, from the programming perspective a seamless transition¹ between on-board memory and external memory. The processor uses a fractional-decimal floating point arithmetic, is pipelined, and has a small dedicated stack. It uses a fixed word size of 24 bits, with the smallest addressable entity being one word² and has a 16-bit address space. There are two sets of floating point registers (each of which can be concatenated to make a 56-bit entity), there are four 24-bit registers, and there is a set of 8 16-bit index registers. Hardware facilities exist to instantiate in-hardware rings and circular queues. There are a total of 68 operands, making this a RISC processor. The instruction set is not orthogonal across the register set, and this is stated in the Motorola literature³ to be due to die size constraints. However, the instruction set is

¹

This is not true if timing is of the essence. All external data space accesses share the same physical port, which can cause serious slowdown and bus contention problems if not carefully handled.

²There are instructions to set, clear and test individual bits within a word.

³Reference 37.

heavily paralleled, with the second instruction usually a move. The processor has numerous vectored interrupts, including a software interrupt (SWI), and appropriate mask registers.

On reset, as implemented in the IR Labs controller, this device expects to find an 8-bit wide EPROM at \$C000 in the program space. An on-board firmware loader will pull 1536 bytes from the EPROM and place it, starting at \$0000, in the program space, and then branch to that address. The program space in the range of \$0000 to \$01FF is internal. Both data spaces are internal in the range of \$0000 to \$00FF. Registers are memory-mapped into one of the data spaces. Figure 6-1⁴, below, shows the architecture of the Motorola 56002 DSP.

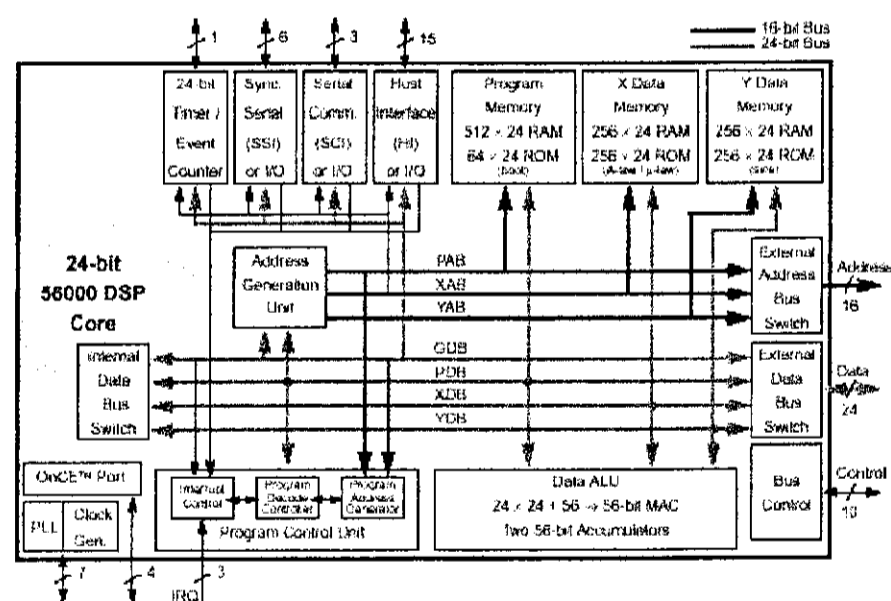


Figure 6-1. Motorola DSP56002 Architecture

⁴From Motorola 56002 DSP User's Manual Figure 1-2. Reference 36

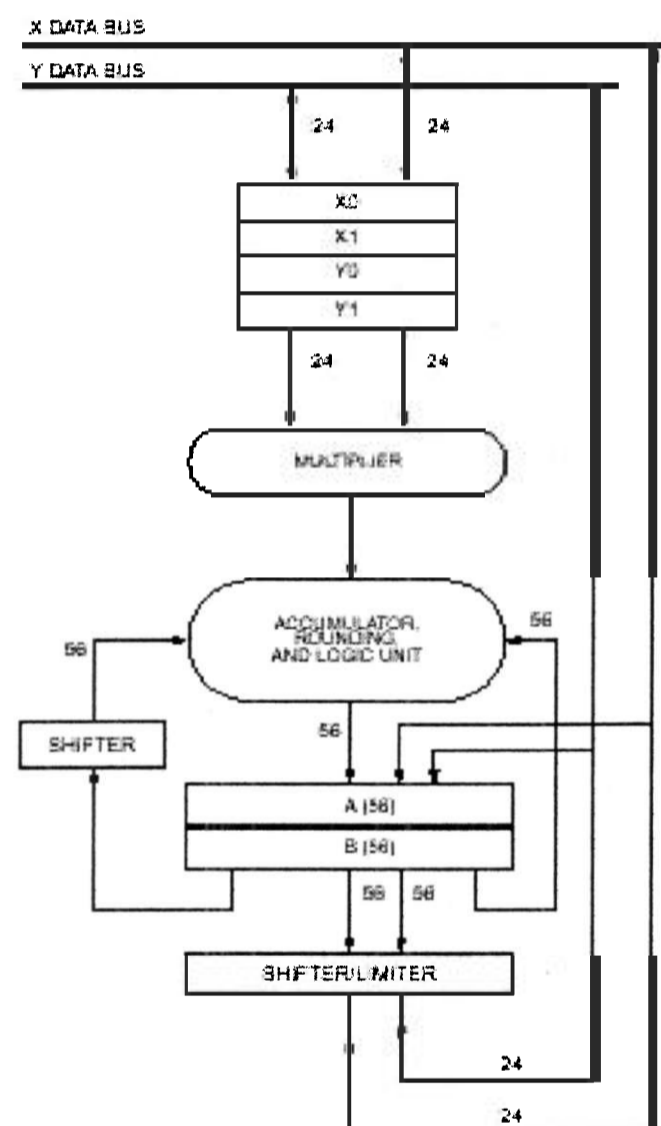


Figure 6-2. Motorola DSP56000 Family Floating Point/Arithmetic Unit⁵

The IR Labs designation for the board upon which the DSP is resident is the “Timing Board”. The timing board is equipped with either a parallel hard-wired connection to the

⁵Motorola DSP56000 Family Manual Figure 3-2, Reference 36.

Spectral Instruments PCI-1826 card or a fibre optic connection to a PCI transceiver⁶. The parallel design incorporates a 16-line parallel interface for fast data transfer and a 9600 baud serial interface⁷ for control transfer. The timing board also is equipped with static RAM for program and data space overflow, and memory-maps various addresses from one of the data spaces onto the backplane in which the board is resident. It is equipped with CPLD logic which has been set up to run various transfers semi-autonomously when properly started by the DSP. There is one on-board timer with interrupt, and a programmable serial I/O port. Figure 6-2 shows the structure of the accumulator/floating point unit.

6.1 Structure of DSP program

When operating, the DSP program consists of two portions. There is the kernel, which includes various service routines including the communication subsystem, the timer driver, the command interpreter, and associated functions. The kernel is a static entity. Then there is the application program, which is brought in from EPROM as an overlay, and which can be replaced at will. The kernel resident on the DSP is a fairly standard small-system real-time executive, written in 56002 assembler and optimized to stay, as far as possible, within a restricted portion of the on-board program memory. By restricting the size of the overlay to use only the balance of the on-board program memory, performance can be optimized.

⁶The version used in this design is the Rev. 3C board with the hard-wired connection.

⁷RS-423.

However, if the functions required of the DSP are sufficiently complex, this may not be possible.

6.1.1 Loader

A portion of the DSP program which is executed only once, and on reset, is the loader. The loader does all the things which must normally be done to initialize an embedded system. This includes properly selecting the memory modes, masking unused interrupts, setting up the timer, and setting up the communication system. Figure 6-3 shows the block diagram for the loader.

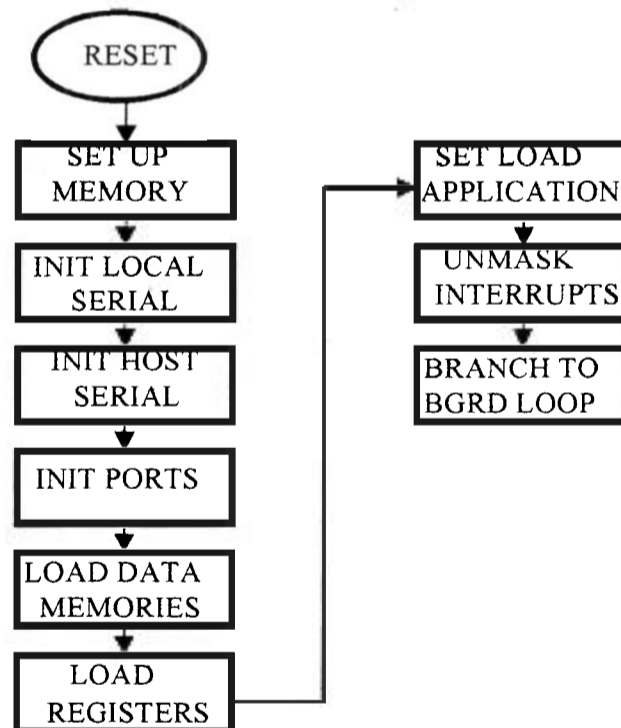


Figure 6-3. Loader Block Diagram

The loader also arranges for the kernel to load the first overlay from EPROM into on-board memory. Since the loader will be executed only once, it is located in the overlay area. Thus, when the application program which is the first overlay is loaded, it is loaded onto the loader. This does not matter however, as the loader has done its job and has passed control to the kernel.

6.1.2 Kernel

The kernel consists of a collection of service routines, including the serial port routines, the command interpreter, the timer driver, and various service routines. The structure of the kernel is centred about a command table of fixed length⁸. When a command is received from the serial port, it is compared against the command table. If a match is found, then a branch to the address associated with that command is performed. If no match is found, then an error message is returned to the host. A block diagram of the kernel is shown overleaf as Figure 6-4.

6.1.2.1 Communication Subsystem

The communication subsystem is an interesting hybrid of interrupt driven tick-tock and direct feed of the serial port. The receive system is driven by a tick-tock as described in Chapter 5. However, the transmit system sends the message one byte at a time and is not

8

The IR Labs code has 24 slots for commands, however this has been increased to 32 slots in the RAO code to allow additional functionality in the application programs.

interrupt driven. It uses the interrupt flag, but only to determine when to load the next byte into the transmit register. It does not return to the background loop between bytes.

The minimum record length that it can recognize as a valid command is two words. The second word must be of a particular format⁹ for the incoming record to be compared against the command.

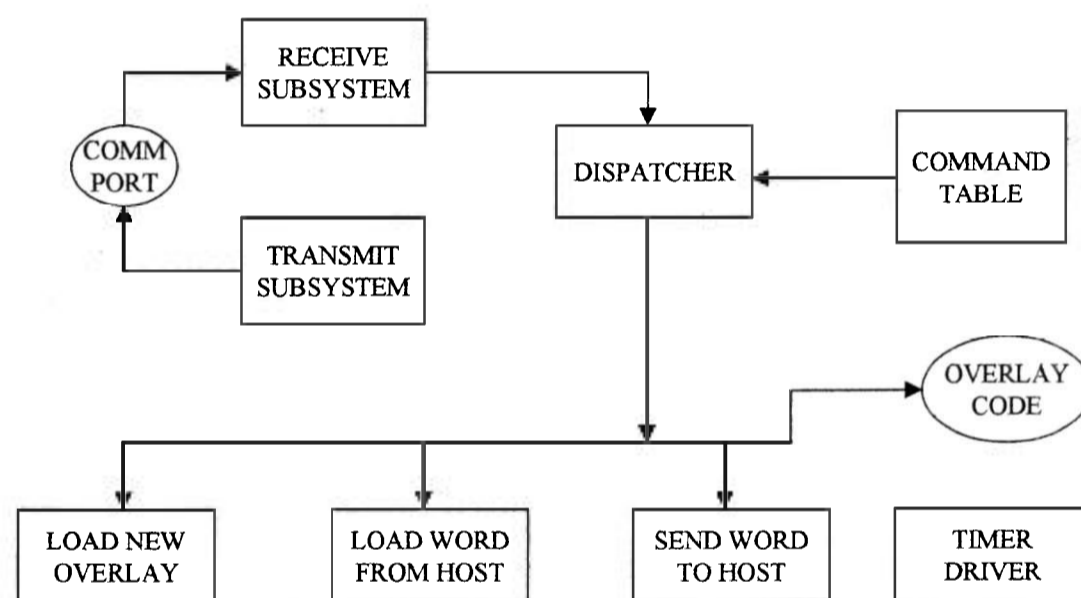


Figure 6-4. Block Structure of Kernel

6.1.2.2 Timer Driver

The hardware timer/counter is loaded such that with a 50 MHz clock it generates ticks at the rate of once per millisecond. This, therefore is the resolution of the timer. This

⁹0x20200d

timer is not used as a clock but is used as an interval timer during exposure, and also to time various wait periods required by mechanical assemblies to move, or for electronics to ramp up.

The timer runs only when it is invoked and is not interrupt driven. A count is loaded into a register, the timer is enabled, and the number of ticks specified by the count is counted out. When this occurs, the routine exits. Thus, the DSP is trapped in this routine for the duration of the count. The only exception is if an event occurs on the serial link. This is monitored by the timer routine, and if a command is received then it will be processed by the dispatcher.

6.1.2.3 Load New Overlay Function

There are two ways to load application programs into the DSP. The first is to load an application program from EPROM under control of the kernel. The second is to download an application program from the host. This section addresses the first function.

The EPROM for this system is a 27C256 EPROM, which is a 32,768 byte device. The loader and kernel take up only 1536 bytes, so there is unused space in the EPROM for other material. Since the EPROM is in the top half of the program space, it can be accessed by the kernel, and can be used to store material other than the loader and kernel. Specifically, it can be used to store one or more application programs.

The kernel code as supplied by IR Labs includes a loader which will load a 272 word application program into the internal program memory, a 16 task command table for this

application program, and 128 words of data to one of the data memories. In the RAO application the kernel has been modified to have a 24 task command table¹⁰, and a 528 word application program, with 128 words of data to be loaded.

Each application is assigned a block of space in the EPROM, and requires 1024 words (3092 bytes) of space. Each application is assigned a number, starting with "1". There is room in the EPROM under this scheme for nine application programs, numbered one through ten, with six missing as it overlays the loader and kernel.

To load a new application program you merely specify the desired application program number, with the load command¹¹ to the kernel, and it will take care of the rest.

Note that the IR Labs software is loaded only to the internal program memory. The RAO loader will load an application of approximately twice the size of that loaded by the IR Labs loader, and the overflow will appear in the static RAM. This does not mean that you are required to have an application program that spills into the static RAM but if you do it will be properly loaded and will execute.¹²

¹⁰

There are 32 slots in the command table but eight of them are taken up by the kernel, leaving 24 for the application program.

¹¹e.g., "LDA 001".

¹²

Due to the fact that all three memories access external memory through the same port, there can be bus contention and access problems. If execution time is a critical consideration then access to external memory should be carefully planned.

6.1.2.4 Load Word

The Load Word function is the second way to load an application program into the DSP. It is a task listed in the command table of the kernel and can be invoked from the host. It will load the word included in the command from the host to the specified address in either data space, the program space, or the EPROM¹³. Repeated sequences of this command can be used to load applications and other data as required¹⁴.

6.1.2.5 Read Word

The Read Word function is the companion to the Load Word function. It is a task listed in the command table of the kernel, to be invoked from the host. It will transmit to the host the contents of the word specified by the address given in the command received from the host. All memories are addressable including the EPROM, although we point out that the EPROM occupies the top half of the program space, so it is accessed by accessing the top half of the program space.

¹³

This requires a byte-erasable EEPROM, which the timing board will support if properly jumpered. However the RAO host program described elsewhere in this document does not support writes to the EEPROM.

¹⁴

However, this requires that the host program be able to interpret the COFF file generated by the linkage editor in order to be able to put everything at the correct addresses and in the correct memories. This feature is planned for the host program and a stub exists in the structure, but writing a COFF interpreter is a major task and was not implemented in the version of the host program discussed in this report, due to time constraints.

6.1.3 Application Program

This discussion concerns the application program for the RAO as derived from the IR Labs supplied application program. This program is application ONE¹⁵ as stored on the EPROM, and is the application program loaded by default on reset. The command table for this application consists of sixteen tasks and is spliced into the command table by the kernel when the application is brought into program memory from the EPROM. There are eight spare slots in the command table, and approximately 260 spare words for expansion as required.

The control requirements of the Rockwell TCM-1000C IR array are fairly simple as compared to other arrays. In the case of the Rockwell array, it is possible to read the array out through a single analog-to-digital converter, and the timing signals use only three lines. One additional line is required to gate power to the preamplifier, and four additional lines are required to control the shutter solenoids.

The readout routines supplied by IR Labs in their application program generate timing signals by outputting bitmaps in accordance with tables¹⁶. The core readout routines have been retained in the RAO application but the driving routines have been modified and

¹⁵See Section 5.1.2.3 concerning application program numbering and layout.

¹⁶

The timing tables in the application code are quite short but contain bit patterns that the DSP puts onto the backplane and which the clock generator board passes on to the IR array. The code takes advantage of the fact that the array is rectangular to run loops to output the bit patterns.

The bit patterns, which appear at regular intervals, comprise the pulse pattern required to read the chip, gate the preamplifier, and to open and close the shutter.

the tables entirely replaced in order to accommodate the needs of the TCM-1000C. The requirements that the power control and the shutter control lines be held constant while the timing signals are generated, but that all of the lines be asserted through the same port, means that masks be kept in memory and logical OR'ed with the timing signals before the timing signals are asserted to the port.. This was not a feature of the IR Labs code.

The IR Labs code also included two continuous-run video modes which transmitted images via the DMA link to the host automatically and regularly. While this code may prove useful at a future date, it is not applicable to the IR camera in its present incarnation as a stare mode instrument, so this code has been excised in its entirety. Approximately 80 percent of the IR Labs code has been replaced, including some new diagnostic code, appropriate to the operation of a stare mode instrument.

Since the RAO application has a specific purpose, a number of details have been hard-coded into the application. Specifically, the application expects the timing board to have the address of 0x0000; that the clock generator board have the address of 0x0002; that the video board has the address of 0x0000; and that only Channel A of the video board is used.

A complete list of the tasks available are shown in Table 6-1, overleaf.

MNEMONIC	FUNCTION
PON	Turn low voltage power on to the timing signals and preamplifier
POF	Turn low voltage power off
SBV	Set clock generator board bias voltages
SB2	Set video board video offset
SGN	Set video board gain
STP	Stop video mode (left in from IR Labs code)
RDA	Read array. Results in DMA transfer.
RRR	Reset array, read, expose, read. Results in DMA transfer.
MRA	Multiple read of array. Results in DMA transfer
ABR	Abort. Stops interval timer and returns.
DON	Immediate return, included to trap bad command.
TST	Returns "TST" to host. Communication test.
SEX	Set exposure timer. Writes to counter for interval timer.
LDW	Load word. Loads word to output port for diagnostic
OSH	Open shutter.
CSH	Close shutter.

Table 6-1. Tasks Available in RAO Application as of October, 2001

6.1.3.1 PON - Power On

The power on command loads the clock generator board DACs, which are serial DACS, using tables which are loaded as part of the application program. The tables supplied by IR Labs have been rewritten to define values and functions appropriate to the RAO application. This command will initialize all DACs, turn on the DC low-voltage supply, wait out all ramp timings, and apply power to the preamplifier. It incorporates the Set Bias Voltage (SBV) task as a subtask. It returns "DON" to the host program.

6.1.3.2 POF - Power Off

The power off command simply shuts off the low-voltage power supply. It returns a “DON” to the host program.

6.1.3.3 SBV - Set Bias Voltage.

The Set Bias Voltage command loads the clock generator board DACs using the RAO tables which are loaded as part of the application program. While this task has an entry in the command table it is not directly accessible to the host program. In the application program it is instantiated as an entry point in the power on (PON) task. A major difference between the IR Labs implementation and the RAO implementation is that the IR Labs implementation uses bias voltage generators on the video board to supply various bias voltages to the IR array. In the RAO implementation these bias voltage generators are not used and the IR array is fed from fixed-output bias voltage generators located in the preamplifier. This command returns a “DON” to the host program.

6.1.3.4 SB2 - Set Video Board Video Offset.

There are two ADC channels on each video board, of which one is used in the RAO application. These channels are each equipped with an input bias voltage offset generator. The purpose of this command is to set the bias voltage. Since in the RAO application only Channel A of Board 0 is used, this task is hard-coded to set only that channel. Also, the set range of this task is restricted to ± 5000 millivolts in order to protect the analog to digital

converter. This command returns a “DON” to the host program.

6.1.3.5 SGN - Set Gain

The video board ADC channels each have a selectable gain amplifier upstream from the ADC. Though the gains are selectable under program control, they are fixed in the sense that you have your choice of four. The available gains are 1.0X, 2.0X, 4.75X and 9.5X. This command returns a “DON” to the host program.

6.1.3.6 STP - Stop Video Mode

This command is a leftover from the original IR Labs code. It has been left in because it sets some flags which may prove important in the future. It is not expected to be executed. This command returns a “DON” to the host program.

6.1.3.7 RDA - Read Array

The Read Array command is a derivative of the original IR Labs code. It has been modified to mate up to the Rockwell TCM-1000C array. This command performs an immediate read of the array. Invoking it will result in a DMA transfer of data from the IR Labs controller to the host. In the RAO application it was retained for possible use as a diagnostic tool, but the version of the host program described in this report does not make use of it. It returns a “DON” to the host program, along with the DMA transfer.

6.1.3.8 RRR - Read, Reset, Read

The RRR command is a derivative of the original IR Labs code. This is the main data acquisition task. It has been modified to mate up to the Rockwell TCM-1000C array. As originally implemented, it would result in a DMA transfer of data from the IR Labs controller to the host for each read, for a total of two transfers. However, the first read has been modified to a clear array and does not result in a DMA transfer. Additional modifications include forcing the shutter closed before the clear, then opening the shutter before the interval timer is invoked and closing the shutter after the interval timer times out and before the read which results in the DMA transfer. A further feature of this task not in the IR Labs code is that when the interval timer is set to zero, the shutter is not opened at all. This allows the acquisition of dark images¹⁷. This command is instantiated as an alternate entry, with a read count of ONE, to the MRA task described below. It returns a "DON" to the host program, along with the DMA transfer.

6.1.3.9 MRA - Multiple Read of Array

The Multiple Read of Array task will read the array the number of times specified in the command received from the host program. The read count is in the word after the command word. This task is instantiated as the principal entry for the main data acquisition task, and the task itself has been modified to meet the needs of the Rockwell TCM-1000C array, as described above. This particular entry, while included in the command table, is not

¹⁷Also called bias images.

directly used by the host program described in this report.

6.1.3.10 ABR - Abort Read

The Abort Read task works only when an exposure (RRR or MRA, described above) is underway. This task works by forcing the interval timer counter to zero, thereby causing the interval timer to return. The DMA transfer which is part of the read array tasks will still occur, but the read array task will not complete. A precursor to this task, with the same mnemonic, is found in the IR Labs supplied code, but works somewhat differently. This task does not return anything to the host, but the read array task will return "DON".

6.1.3.11 DON - Done

This entry in the command table is actually a trap for strings which might be generated by the host program in error. It is in the IR Labs code and was retained in the RAO code. It branches directly to the background loop and returns nothing to the host program.

6.1.3.12 TST - Communication Test

This task has no counterpart in the IR Labs supplied code. It was originally inserted into the application program as a diagnostic, but is useful in testing whether the serial data link is operational, and if the controller is operational. It echoes "TST" back to the host via the serial link. It does not return "DON".

6.1.3.13 SEX - Set Exposure Time

This task has no counterpart in the IR Labs supplied code. Requesting SEX will cause the word following the command word in the DSP input buffer to be loaded into the timer target time counter. The timer is not started however. This command returns a "DON" to the host program.

6.1.3.14 LDW - Load Word

The Load Word task has no counterpart in the IR Labs supplied code. This task was originally inserted into the application program as a diagnostic tool, and is retained as such. Invoking this task will result in the word following the command word in the DSP input buffer to be loaded to the SS (Switch State) lines on the backplane. It returns a "DON" to the host program.

6.1.3.15 OSH - Open Shutter

The Open Shutter task is broken into two parts. The task started by the entry in the command table merely calls a subroutine, and it is the subroutine which does the work. The table was set up this way because it makes the subroutine available for inclusion in other tasks, specifically the RRR task (described above). The task uses a mask which can be logical OR'ed with control words that are put out by array read routines, and sets bits in this

mask to force the solenoids¹⁸ to move the shutter, and to hold the shutter. The task works by setting and clearing the appropriate bits in the mask, and then loading this mask to the output port. This task will result in hardware movement, and will return a “DON” to the host program.

6.1.3.16 CSH - Close Shutter

The Open Shutter task is broken into two parts. The task started by the entry in the command table merely calls a subroutine, and it is the subroutine which does the work. This routine operates in a manner similar to that of the Open Shutter routine, described above. This task will result in hardware movement, and will return a “DON” to the host program.

¹⁸

There are two solenoids, one for open shutter and one for close shutter. Because the camera can be at any orientation, it is not sufficient that the shutter be moved to one position and then the solenoid be depowered. A small current through the solenoid is required as a holding current. The solenoid mask must reflect all this.

CHAPTER SEVEN

HOST PROGRAM

There are three kinds of lies: lies, damn lies, and statistics.

Benjamin Disraeli, 1874

CHAPTER 7.

HOST PROGRAM

7.0 Introduction

The host program is a control program written in ANSI C to run under Microsoft Windows 9.x or better. There are roughly 45 files associated with the source code, including test images, GUI files, DLLs, libraries, include files, source code files, and ancillary items. The purpose of the host program is to provide a user interface for the operation of the infrared camera, to provide the capability of controlling the filter wheel assembly within the camera, and to provide image storage capability with FITS format files. The host program communicates with the IR Labs array controller via a proprietary channel utilizing a PCI card. A development feature (white rat¹) has been deliberately left in the release code to facilitate modification of the host program as required. The host program is dependent upon a number of external libraries and packages. These libraries and packages will be discussed in this chapter. The source code for the host program is included in this report as Appendix D.

7.1 Program Structure

The host program was written using a National Instruments compiler and GUI tools and utilizes libraries unique to this compiler family. The GUI tools are capable of

¹
See Section 7.5

instantiating callback function capability but they also incorporate a function² to interrogate the user interface outside of a callback. There are several direct callback functions in the host program. However, these are used only for menu bar callbacks and some timer functions. The user interface is coupled to the program through the GUI interrogation function. In order to avoid having global variables, an object (typedef DATABLOCK³) is defined during initialization, as well as a pointer to this object which is used by the host program as the handle to the object. All items which are shared by all functions are within this object and the pointer to this object is passed during execution. A high-level block diagram of the background loop, showing program flow and

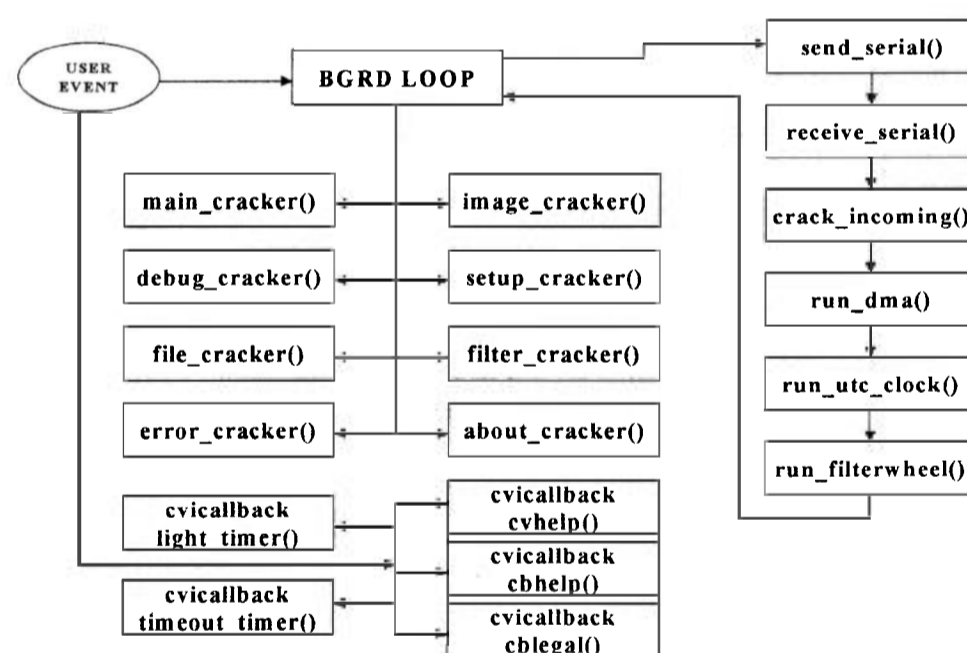


Figure 7-1. Host Program Background Loop

²GetUserEvent(), Reference 48.

³See Appendix D.

direct callbacks, is shown as Figure 7-1.

7.1.1 Structure of Graphic User Interface

The Graphic User Interface (GUI) consists of a collection of windows which were developed using the appropriate National Instruments tools within the compiler. When such a window is developed and saved, the window itself will be saved as a file with the extension UIR, and an include file will be generated as well. This include file must be referenced in the host program for the host program to be able to access objects within the window. One window is the parent window, all other windows are child windows of the

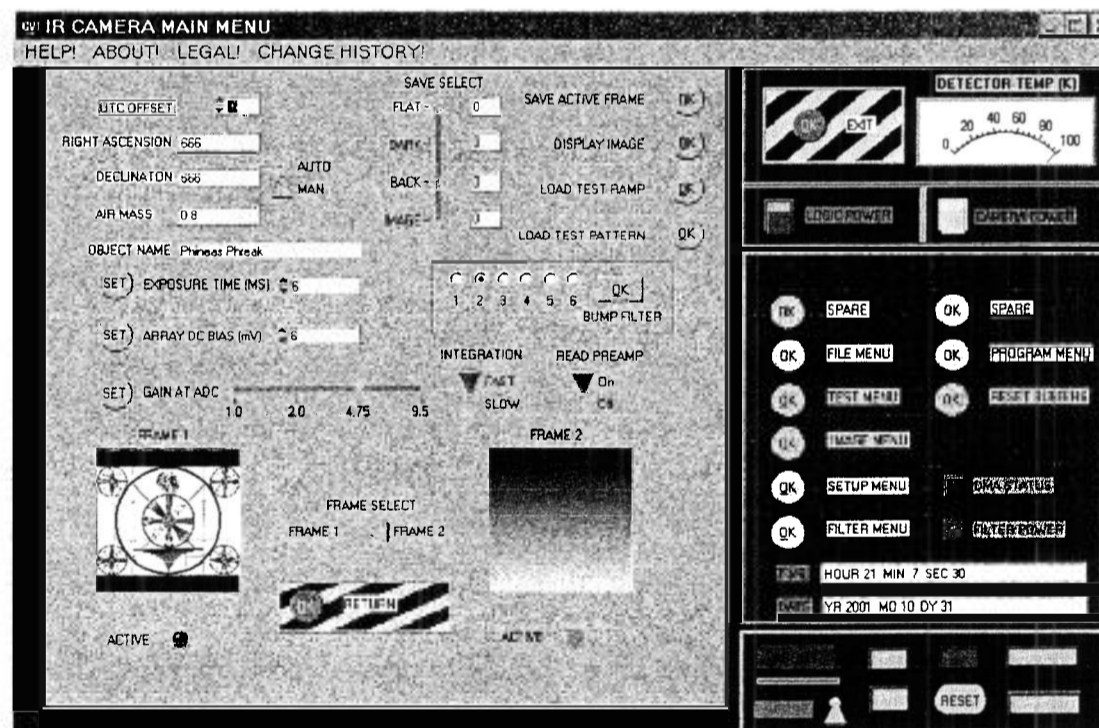


Figure 7-2. Typical Control Window

parent. The image control window, shown in Figure 7-2, is a typical control window.

Each window has a handle associated with it, and the value of that handle is dependent upon the sequence in which the windows are created. Because the values of the handle are dependent upon the order in which the windows are created, there are define statements within the host program which must be synchronized with the window setup routine to ensure proper operation of the host program. The windows are not displayed until a function is invoked to display that window explicitly. Further, a window may be removed from the display without removing it from memory by invocation of the appropriate function.

7.1.2 Structure of the Background Loop

After initialization, the host program drops into a background loop (forever loop), which it can leave only if there is an error⁴ or the operator requests the program to exit. The background loop performs several tasks and then interrogates the GUI. If there is a user event on the GUI, it obtains the handle of the window generating the GUI. It then executes a switch statement on this handle to select the correct handler for the window generating the user event. It is this switch statement which requires the define statements referenced in the previous section to be synchronized with the order in which the windows are created. Each case within the switch statement corresponds to a different window.

⁴Error code meanings are given in Appendix E.

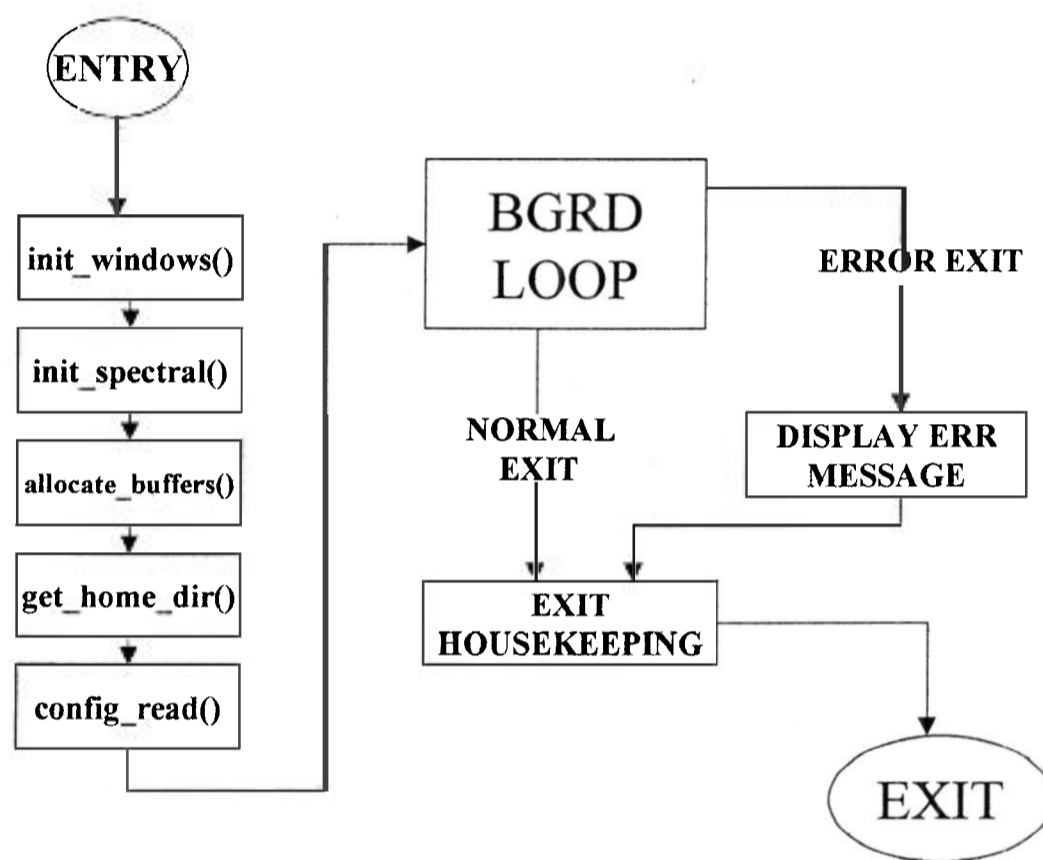


Figure 7-3. Program Flow On Initialization

Within the case statement, a function is invoked which is specific to the window which has generated the user event. This function receives the identity of the object in the window which generated the user event and executes a switch statement on this identity. Finally, the case statements for this last switch contain calls to the functions which actually do the work.

7.1.3 Structure of Communication Subsystem

The communication subsystem is built around a Spectral Instruments⁵ PCI-1826 interface card. This card has a UART on board and also a level shifter to generate RS-422/423 differential signals. It also has an on-board DMA controller which receives data from the IR Labs controller. The serial link is a control channel but does not pass image data from the camera. The DMA channel is a 16-bit parallel channel but does not pass control data.

7.1.3.1 Interface to Spectral Instruments PCI-1826 card.

The Spectral Instruments card is a PCI card capable of seizing a DMA channel and which can also generate interrupts. In a Windows environment it requires a device driver (VxD) which must be present in the C:\WINDOWS\SYSTEM directory. This driver dynamically loads and links to the card on host program startup and unloads on host program exit. The interface between the device driver and the host program is via a DLL, and this DLL can be anywhere in the path on the host machine. The installation program however will install the DLL with the executable in the host program directory.

⁵

All software and documentation will be found on the associated CD-ROM as listed in Appendix H. The material associated with the PCI-1826 is copyright (c) Spectral Instruments and is the property of the copyright holders. Permission to duplicate must be obtained from Spectral Instruments Corporation.

7.1.3.2 Serial Communication Channel

The serial communication channel is locked at 9600 baud, 8N1. This can be changed in the host program by changing the appropriate parameters in the include file (svid.h) and recompiling, however the code in the IR Labs controller must be changed. The serial communication channel is set up as a tick-tock, as described in Section 4.4.2.1 of this paper. The communication system on the host side uses single buffers, however there is an underlying layer provided by the DLL which allows latency, so a multiple buffer scheme is not necessary.

All functions called by the background loop (as a result of a user event which generates strings for transmission to the IR Labs controller) load the transmit buffer and then set a flag which indicates that there is a message to be sent. The background loop then strobes the contents of the buffer out, resets all pointers, and clears the flag.

The IR Labs controller will respond to a transmitted string with one of a set of responses. If the controller does not respond within ten seconds then an indicator is displayed on the GUI to show that a communication failure has occurred. If the controller responds but with an error message, an indicator is displayed to indicate this as well. If the controller responds with the expected message, an indicator is displayed to indicate that the handshake occurred successfully and all timers are shut off.

There are very few strings which the host can send to the controller which will result in other than one of the above mentioned set of responses. For this reason, the message cracker for strings received from the controller is somewhat rudimentary.

7.1.3.3 DMA Channel

The DMA channel is reserved for data transfer. The DLL⁶ has a set of functions for dealing explicitly with the DMA channel, including a function which tells the DMA controller the address of the buffer into which it should drop the transferred data. There also is a function which can be used to determine when the transfer is completed, and this function is used by a function called by the background loop to monitor the status of the transfer and to set the colour of an indicator in accordance with the status of the transfer.

This channel is double buffered within the host program but there is no automatic select between the buffers. Rather, in one of the windows in the GUI there are two display objects⁷. Between them there is a bat-handle switch to select the desired display object. Each display object is associated with a DMA buffer, and selecting the display object also selects the DMA buffer. However, the contents of the buffer are not displayed in the display object until the user requests that this be done.

Having the DMA channel double buffered allows the user to obtain one image, then switch to the other buffer and work there while using the first image as a reference. A time stamp is associated with each buffer. This time stamp is the time that the image was uploaded from the controller.

⁶

For information on how to access a VXD driver via a DLL see Hazzah (Reference 8).

⁷

Figure 5-2.

7.1.4 Real-Time Clock

The host program is designed to run on a Pentium II or better PC with a real-time clock. The development platform has access to the Internet and has a utility⁸ which runs periodically to synchronize the internal PC clock with a national time standard. The utility takes into account the transmission delay and claims to be able to ensure an accuracy of within 100 milliseconds. The host program uses the internal PC clock and information supplied by the user to offset the value of the clock to recover UTC. It then displays the UTC time and date on the GUI, and also uses this information as the timestamp when an image upload occurs from the IR Labs controller.

7.1.5 Image Storage

The host program will store images on request. It will change to a working directory, which it can create if necessary. The user can specify the name of the working directory, including drive letter, from the GUI. The host program will then create within the working directory four subdirectories, to wit, FLATS, DARKS, IMAGES, BACKS. There is a four-position select in the GUI which selects which of these subdirectories to which the image being saved is spooled. The host program assigns file names sequentially, but it is up to the user to maintain proper logs so that he knows which image is which.

All files are in FITS format, and all files are tagged with header information as supplied by the user via the GUI.

⁸Automachron, which is available from One Guy Coding for download as freeware.

7.1.6 Filter Wheel Control

The filter wheel is controlled via a ACCESS® DIO-128© parallel interface card. This card uses several Intel 8255 chips and is a PCI card. The card requires a VxD and the VxD communicates with the host program via a DLL. The DIO-128 does not control the filter wheel directly. Rather, it communicates with an in-house designed module which receives information from the DIO-128 and directly controls the stepper motor which turns the filter wheel. This module also returns to the DIO-128 an indication of the filter wheel location, and also temperature information (in the form of packed BCD) concerning the temperature of the Rockwell TCM-1000C IR array. The temperature information is displayed on the GUI in degrees K. The filter wheel control also houses the power monitor for the array preamplifier, which is used to indicate to the user that the preamplifier does indeed have power.

7.2 Host Program Auxiliary Systems

The host program utilizes a number of services available from the Windows 9x environment. Among these are the use of an HTML-capable browser, which is used by the host program for displaying an HTML-based on-line help facility.

7.2.1 Help Subsystem

The host program is equipped with on-line help feature which can be started by clicking on the task bar at the top of the main window. This help feature is written in

HTML⁹ and will allow the user to understand and use the various features of the host program. Other features¹⁰ are available from the taskbar.

7.3 Dynamic Linked Libraries

The National Instruments CVI C compiler uses the Microsoft API call structure. This call structure has been superseded by another Microsoft call structure automatically generated by the Microsoft Visual C++ Version 5.0 compiler, and higher versions. In order to use DLLs generated by the Microsoft compiler or which were obtained from other sources, a library (LIB) file must be generated to match the the particular DLL, and then included in the fileset used by the National Instruments compiler.

7.3.1 Generating LIB files From DLL Files

National Instruments is aware of the problem of incompatibility between the two forms of Microsoft call structures, and has engineered into their tools features which allow for the generation of LIB files from third-party DLL files. In order to do this it is necessary to make a separate project using the National Instruments project manager, and to acquire

⁹

The HTML source code for the help function is found in Appendix G.

¹⁰

ABOUT, LEGAL, and CHANGE HISTORY. Clicking on ABOUT will display ownership and version information. Clicking on LEGAL will display the conditions of usage for this program. Clicking on CHANGE HISTORY will display the history of the changes for this program. The HTML source code for these features is found in Appendix G.

or write an include (.h) file which has the correct declaratives (e.g., void __declspec(dllexport) foobar(int);) for all functions within the DLL. Then, by following the procedure given in the documentation it will be possible to obtain a LIB file for the CVI compiler. This LIB file must be included in the fileset used by the National Instruments compiler during the build.

7.3.2 FITS Library

The FITS library used was downloaded from NASA Office of Science and Technology¹¹, and was originally written for the Sun Solaris platform. Solaris is a unix variant and does not use DLL files so, although this library was written in ANSI C, it was not set up for use as a DLL. Further, it employs library functions which are part of the ANSI C specification for unix platforms but which are not included in the National Instruments compiler library. These library functions are, however, part of the Microsoft Visual C++ run-time library.

The solution to this dilemma is to compile the FITS library as a DLL using the Microsoft Visual C++ compiler¹². By doing so, a stand-alone executable incorporating all required functions can be obtained.

¹¹<http://fits.gsfc.nasa.gov>.

¹²

If this compiler is used to generate a DLL, that DLL will have all external references resolved, and will be a true executable. This gets around the problems with the linkage editor.

However, the FITS library, having been originally written for a platform for which the concept of a DLL does not pertain, does not have the proper declaratives in its function specifications and header files. Due to the size of the library (750K bytes of source code) it was deemed not practical to convert all functions to have the proper declaratives for use as a DLL.

The solution was to utilize wrapper functions. At some slight performance cost (one more stack frame and some clock cycles lost on another call) wrapper functions which do incorporate the proper declaratives were added to the FITS library build. The include file specifying the wrapper functions was then written and the whole library was compiled as a DLL. The LIB file was obtained as specified in Section 7.3.1.

Error codes for errors specific to the FITS package are given in Appendix F.

7.4 ACCESS I/O DIO-128

The ACCESS I/O DIO-128 is a parallel I/O card equipped with 5 Intel 8255 parallel output devices. The driver for the board must be installed by running the software obtained from ACCESS via their CD. This installation software will find the card and return information needed by the host program, including the address of the port. This address must be loaded into the host program setup menu. However, once loaded it can be stored in the CONFIG.DAT file and need not be reloaded.

The base address returned by the installation software will be only for the first 8255

output device. Each 8255 output device¹³ has four ports, of which three are user configurable general-purpose I/O, the fourth being the control port for the first three ports. Since there are five 8255 devices on each DIO-128, the addresses of the ports are incremented in multiples of 4 to cover all the devices on the board.

The DIO-128 is used by the host program to transmit data to the filter wheel controller and to obtain filter wheel position, detector temperature information, and preamplifier power status. To do this requires the use of only one 8255 device and at that there are a few spare lines, so there is room for expansion for other uses.

7.5 White Rat

The white rat¹⁴ is a debugging feature which is deliberately left in the release code for the host program to facilitate code development for both the host program and for the IR Labs Motorola DSP. It requires a dumb terminal¹⁵ attached to COM 1, and set to 9600 baud, 8 data bits, one stop bit, no parity (9600 8N1).

There is a switch in the DEBUG MENU screen that enables the white rat. When the white rat is enabled it will echo all traffic both to the IR Labs controller and from the IR

¹³Programming instructions for this part are freely available on the Internet.

¹⁴

In naval shipboard sound-powered telephone systems, officers will clip on to the circuits with an amplifier and speaker so they can hear the traffic on the system without alerting personnel using the system that they are listening in. These devices are known as "White Rats".

¹⁵Or anything that looks like a dumb terminal.

Labs controller to the dumb terminal. The software is not set up to display characters, rather it will display the hexadecimal code for the characters transmitted.

7.6 Errors

There are three possible types of errors which can occur. These are fatal error condition, correctable error condition, and error arising from within the FITS library.

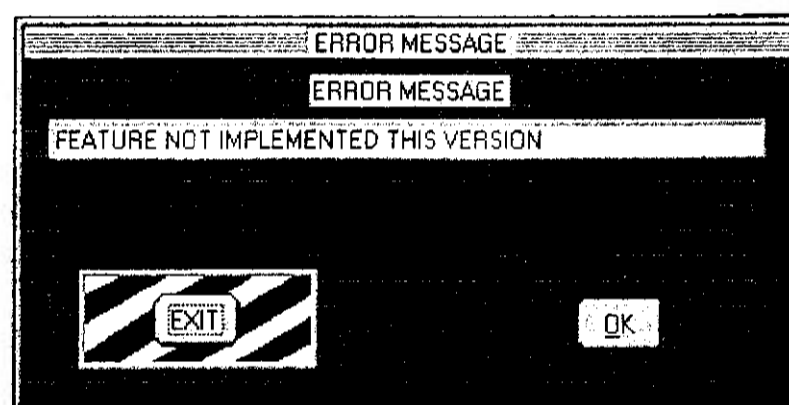


Figure 7-4. Error Screen

7.6.1 Fatal Error.

If the host program encounters a fatal error then the program will immediately abort and display a message to SYSERR¹⁶ stating the application screen number and the control number which generated the error. These errors are caused by coding errors or bugs and

¹⁶

If SYSERR is not available the CVI run-time package will open a window for SYSERR.

should be reported as displayed to the author¹⁷ of the code.

7.6.2 Correctable Error

If the host program encounters a correctable error condition¹⁸ during operation it will display a special screen (such as that seen in Figure 7-4) which states the error and gives the user the option of continuing (by clicking on OK) or of exiting immediately. If the user opts to continue, she should be aware that performance will be degraded by the condition which caused the error.

7.6.3 FITS Library Error

As stated above, the FITS library is a canned library obtained from NASA and written for another platform. The encapsulation around this library and the integration with the host program is believed to be fairly good, however there exists the possibility that the FITS library will generate an error condition. If this occurs, the error will be displayed on the same screen as that shown in Figure 7-4, with the notation FITS ERROR XXXX, where XXXX is the number internal to the FITS library and which denotes the nature of the error. The error codes obtained with the library are reproduced in Appendix F.

¹⁷

Try goanna@shaw.ca or goanna@postmaster.co.uk.

¹⁸

Examples of error conditions include the Spectral Instruments card not present, critical files not available, and errors in the configuration file data.

CHAPTER EIGHT**CONCLUSION**

Dulce et decorum est pro patria mori

Horace

Like hell.

Country Joe McDonald

CHAPTER 8.

CONCLUSION

8.0 Overview

The objective of the research was to build an infra-red camera, image objects with the camera, and to investigate the characteristics of these objects. Preliminary work at the RAO using the 41 cm telescope indicated that it would be ideal for photometry on extended objects. One particularly appealing task was the possibility of doing temperature maps of Seyfert galaxies¹ and other objects with active galactic nuclei. About three months of work went into this effort, using both the 41 cm telescope and the ARCT, and the feasibility of such an effort has been shown². This investigation was run in parallel with the task of assembling the components for the IR camera.

Unfortunately, due to the complexity of the actual assembly of the camera and support systems as well as the sequence of events described below, it was not possible to do any photometry using the camera.

¹

One particularly appealing object is NGC 4631. This object covers three arc-minutes, and covers an entire plate on the 41 cm telescope. To image this object with the ARCT would be difficult at best given the present condition of the ARCT.

²

There are some considerations concerning the mounting of the telescope at the ARCT. These considerations will not be addressed in this document.

8.1 Status As Of This Writing

As of this writing (late October, 2001) the IR camera has yet to see first light. The camera is at this writing completely assembled, save some wiring on the dewar plug and the cabling between the controller and the preamplifier assembly.

The filters that are in place are not what had been hoped for. The filters now in place are not filters that are in the Johnson-Cousins system. They have narrow passbands, and are designed to remove, as far as is possible, the effects of water vapour in the atmosphere. While this is very commendable, they have not been properly characterized, and their response as a function of water vapour relative to standard filters is not known³. After some difficulty in placing the order, standard filters were ordered but had not been received by the time this report was written.

The software for the IR Labs controller is patterned on the software as supplied by IR Labs for use with the PICNIC array, but has been completely rewritten for the TCM-1000C array. It is known that the software works, but it has as yet to be tested against the array.

The software for the host is operational and has been tested against the IR Labs controller. It appears to be fully functional.

3

The performance of the narrow-band filters is expected to be significantly better than the Johnson-Cousins filters.

8.2 There And Back Again⁴

The original job of work was described as “Do a little soldering, and you can do some observation and have a nice little thesis⁵”. What was not mentioned in this description was that the IR spectrometer as well as the IR camera were, at the time the job was started, a collection of dusty parts in a disused section of the RAO laboratory. The original intent was to build an IR spectrometer using an Aerojet linear array. As this instrument would have been suitable for the study of point objects it would have mated well with the ARCT as it stands. Unfortunately, due to the history⁶ of the array, this effort was deemed not feasible early on, and the focus was shifted to the Rockwell IR array.

With attention shifted to the Rockwell TCM-1000C, it seemed important to determine the suitability of such a camera at the RAO. This resulted in many nights⁷ at the 41 cm telescope doing imaging, and at the 1.8 metre telescope doing IR photometry, to determine if the construction of an IR camera using the Rockwell array is a reasonable

⁴

Section title from J. R. R. Tolkien, “The Hobbit”.

⁵

E. F. Milone, June 2000.

⁶

Without going into the intricate details of what happened when the author attempted to get technical information from the manufacturer, suffice it to say that the array is an artifact of the Cold War, is on the list of materials which may not leave the US or be in the hands of non-US citizens. They want it back, never mind that it is 20 years old.

⁷

Dr. David Fry, Fred Babott, Larry Harding, and the author in attendance, in various combinations at various times.

proposition. The data derived shows that the sky, while marginal in the M band, does not make observation in this band out of the question, and in the H, K, and L bands the seeing can be quite good. The data also indicates that with a field of view of 55 arc-seconds, in order to use the IR camera on the ARCT it would be necessary to mosaic the images, with ramifications discussed elsewhere in this report. However, an imaging IR camera would give the RAO the capability of studying extended objects in the medium wave IR, which heretofore it did not have. Accordingly the decision was made to press on with the development of the camera.

8.2.1 The Controller Incident

Early in the work with the Rockwell array, it was realized that we did not have all the pieces⁸. This led to discussions on how to obtain a controller, and various proposals were put forth, including building a controller. Investigation of the possibility of building our own controller led to numerous discussions concerning the features desirable in such a controller, and several times concerns were voiced as to whether the controller was desirable at all. Eventually, a preliminary design based on a Texas Instruments TMS320C30 was developed by the author, with the objective of the design being able to do the job at a low

⁸

Actually, this was not true. One of us (F. Babott) had realized some years earlier, when the array was first received from IR Labs, that we did not have all the pieces. At that time he designed a controller and a preamplifier, but these pieces were never built.

cost, yet having sufficient headroom in the design to allow for “function creep”⁹.

UNIVERSITY OF CALGARY

2-5 MICRON CAMERA PROJECT

- WHY THIS PIECE IS NECESSARY

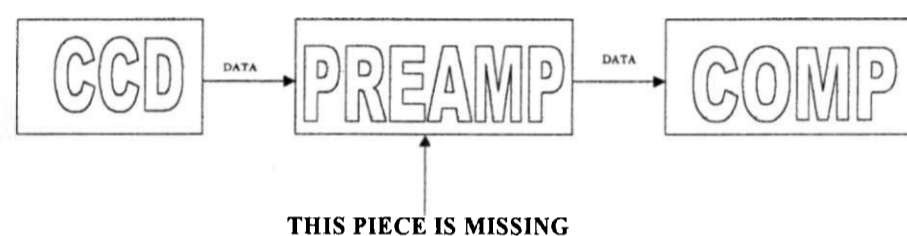


Figure 8-1. Why the Preamplifier is Necessary¹⁰

It was indicated that the cost estimate for the controller was not reasonable, and the author offered to fund the construction of the controller out of her own pocket. This offer was declined.

Eventually a controller was obtained at no cost from IR Labs. The controller was a design prototype, and the documentation available from IR Labs did not match the hardware.

⁹

See Figures 8-2 and 8-3, overleaf.

¹⁰

Figures 8-1, 8-2, and 8-3 are from a presentation given in July, 2000 concerning the need for a controller and preamplifier.

UNIVERSITY OF CALGARY

2 - 5 MICRON CAMERA PROJECT

• **OPTION 1 -
MINIMUM
SYSTEM**

- COST: \$800USD
- CAPABILITIES:
 - MINIMAL
 - CANNOT BE EXPANDED
 - RISK OF UNDERDESIGN

• **OPTION 2 -
FULL
SYSTEM**

- COST: \$1200USD
- CAPABILITIES:
 - INCLUDES DSP
 - CAN HANDLE COMPLEX TASKS
 - HEADROOM FOR FUNCTION CREEP

Fig 8-2. Cost Estimate of Proposed In-House Controller Design

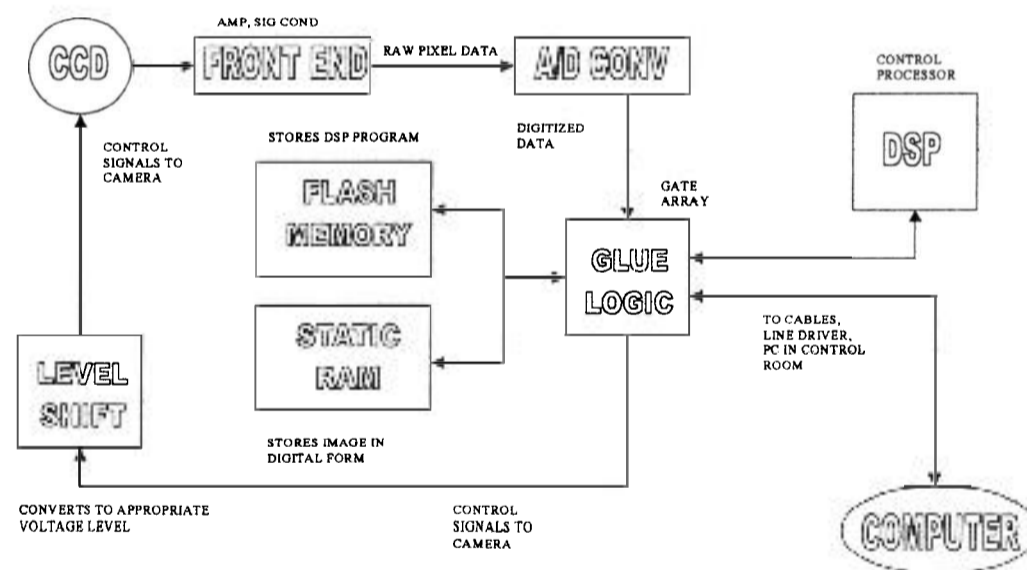


Figure 8-3. Block Diagram, Proposed TMS320C30 Based Controller

However, this difficulty was overcome and in May of 2001 a functioning array controller was in-house.

While all this was going on, Fred Babott and the author of this thesis analyzed the Rockwell IR array and designed support electronics, which Fred Babott subsequently built.

8.2.2 The Computer Factory

In September, 2000 the need was recognized for a computer for software development and to serve as a platform for integrating the various components of the IR camera. A computer was in fact purchased, but it became the data acquisition computer at the ARCT and was not available for use as a development platform. On January 3rd 2001, therefore, the author gathered up the contents of her computer graveyard, scavenged other bits, added new components¹¹, commandeered a laboratory, and started building computers. She built four, altogether¹².

¹¹

The out-of-pocket expense, exclusive of what was pulled out of the graveyard, was approximately \$1500.00CDN. Additional software licence expenditures of approximately \$8000USD were committed to the project. These figures are disputed however documentation exists showing that they are approximately correct. The figure for the software also includes the purchase price for the support tools for the DSP56002 microprocessor in the IR Labs controller.

¹²

Two were traded as a quid-quo-pro for laboratory space. The third (glimdrop) serves as a file transfer node, and the fourth (abscam) is the development platform. Glimdrop and Abscam are the computer names.

8.2.3 Problems with the IR Labs Controller

As received, the controller did not work. After the author read, sorted and digested the available documentation, she stripped out the IR Labs controller, removing all boards except for the timing board. Work with the timing board indicated that it was defective, although this was not readily apparent¹³. When the fact that the timing board was defective was finally determined, it was returned to IR Labs for replacement¹⁴. The replacement board is of a later version, for which correct documentation is available.

8.2.4 Problems with the Spectral Instruments 1826 Interface Card

As it turns out, there was nothing wrong with the interface card, and the documentation was correct. However, the documentation was for a 16-bit MS-DOS device driver, and as the host computer runs under Windows 98, this device driver will not work¹⁵. It was therefore necessary to contact Spectral Instruments, obtain their VxD driver (written in Visual Basic), and to write an interface DLL. This was done, and at that point the card became accessible to software written at the University of Calgary, with the result that it was possible to communicate with the IR Labs controller.

¹³

The DSP56002 would do everything except assert the /WR strobe.

¹⁴

Since the controller was “loaned” to the University of Calgary by IR Labs, the timing board was swapped as a no-cost exchange by IR Labs. There is an invoice however which indicates that the University did in fact pay \$2500USD for the replacement board

¹⁵

Except if it is run in DOS mode, which is not an acceptable option.

8.2.5 Problems with the IR Labs Host Program

The IR Labs host program was written in 1996 and was intended to run on a PC running under MS-DOS only and equipped with a 640 by 480 resolution VGA video adapter card only¹⁶. In order to run the host program under Windows 98, it had to be completely rewritten. It was also intended to control a completely different instrument, and had features completely inappropriate to what was needed.

That the rewrite had to be done in any event to accommodate the Spectral Instruments driver only forced the issue. The result is that the host program presented in this report is a complete rewrite of the original IR Labs host program, comprises some 4000 lines of code, and took approximately four months to develop.

8.2.6 Problems with the DSP56002 Software

The DSP software supplied by IR Labs was not for the version of the timing board supplied, and it was for a completely different array. It was therefore necessary to analyse the supplied software, strip out the portions which were not appropriate for the hardware available, and then to rewrite the DSP software to suit the University of Calgary application. This required an estimated 80 percent rewrite, and while the code described in this report is patterned on the code originally supplied by IR Labs, it is substantially different. Approximately 1000 lines of code were rewritten.

¹⁶

And accept no substitutes. The IR Labs host program will not work correctly with an AGP video adapter card.

8.3 Recommended Work

Aside from completing the assembly of the camera¹⁷ and an end-to-end test of the entire system, the host software should be augmented to allow automatic operation. The present software is designed for full manual operation and is not capable of taking sequences of images with filter rotation between the images. This can be done with a moderate degree of effort and without disturbing the interface between the host computer and the IR Labs controller.

The host software was developed on a Windows computer using a 1024 by 764 resolution monitor¹⁸ and the font size on the development platform was set on SMALL. While this may seem to be a minor point, the software must be tested against the actual target platform in order to ensure that there are no problems with screen size or font size, and that all virtual controls in the host software work correctly on the target platform¹⁹.

There has been some discussion concerning the user interface for the host program. Current thinking for the automated version is to include a game controller such as is found on a Sony PlayStation²⁰. This approach has yet to be explored.

¹⁷

Expected to be complete by New Year, 2002

¹⁸

Actually, a 1024 x 768 window on a 19-inch 1600 x 1200 monitor.

¹⁹

These are standard problems when porting software from a development system to a target system, and there are utilities in the development compiler to deal with these problems. But they are issues which must be addressed.

²⁰

This controller will not mate with the PC however Microsoft makes a USB clone of the

The platform originally intended as the target platform is equipped with an ZIP²¹ drive to facilitate software transfers. However with the uncertainty concerning the actual place of use of the IR camera this has become moot, and alternate delivery vehicles may be necessary²².

8.4 Summary

The IR camera was intended as an instrument for the ARCT, and the investigation surrounding the construction of the camera has highlighted several issues concerning the use of this camera on the ARCT.

While the camera was intended to be a final product, it has also served as a pilot project for other efforts. These include the possibility of mounting another, “store-bought” IR camera with extended sensitivity, and the possibility of upgrades to the RAO. Regardless of the merits of the camera described in this report, the RAO now has the expertise required

Sony controller that will mate with the PC. The Microsoft controller has numerous buttons and also two thumb-operated joysticks for analog input and cursor positioning. To make use of a game controller would require assembling another DLL using the Microsoft Visual C++ compiler, but opens up possibilities concerning ease of object selection and positioning of the telescope.

(Update, 2 Dec 2001). A DLL for the game controller has been made from code extracted from the Microsoft Visual C++ compiler. There are other tasks with higher priority.

²¹

IOMEGA 100 MB ZIP drive.

²²

A CD burner is available if needed.

to install, service, and operate infrared cameras.

REFERENCES

- 1 Bach, Maurice J. “The Design of the Unix Operating System”, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1986
- 2 Barron, D. W. “Assemblers and Loaders”, American Elsevier, Inc., New York, 1969
- 3 Franklin, Gene F. and Powell, J. David, “Digital Control of Dynamic Systems”, Addison-Wesley Publishing Company, Menlo Park, CA., 1980
- 4 Gircys, Gintaras R. “Understanding and Using COFF”, O’Reilly & Associates, Inc., Sebastopol, CA., 1988
- 5 Graham, Ian S., “The HTML Sourcebook, 2nd Ed.”, John Wiley & Sons, Inc., New York, 1996
- 6 Gomez, Martin, “Embedded State Machine Implementation”, Embedded Systems Magazine, Dec. 2000

- 7 Haisch, Karl E., Lada, Elizabeth A., and Lada, Charles J., "A Near-Infrared L-Band survey of the Young Embedded Cluster 2024", *Astro. Journal* 120:1396-1409, 2000 Dec.
- 8 Hazzah, Karen, "Writing Windows VxDs and Device Drivers: Programming Secrets for Virtual Device Drivers", CMP Books, Lawrence, KS, 1997
- 9 HEASARC, "CFITSIO User's Guide: An Interface to FITS Format Files for C Programmers, Version 2.2", HEASARC, Code 662, Greenbelt, MD., 2001
- 10 Higgins, Richard J., "Digital Signal Processing in VLSI", Prentice-Hall, Englewood Cliffs, NJ., 1990
- 11 Infrared Labs DSP Software, dtd 19 Jan 1998 (unpublished)
- 12 Infrared Labs Generation II Timing Board, dtd 25 May 1998 (unpublished)
- 13 Infrared Labs System Description (undated) (unpublished)
- 14 Infrared Labs TCM-1000C Support Electronics, dtd October 1989, internal report (unpublished)
- 15 Infrared Labs Video Processor Board Description, dtd 16 Nov 1998 (unpublished)

- 16 IBM Corporation, "System/360 DOS/VSE System Programmer's Guide", IBM Press, New York, 1969.
- 17 IBM Corporation, "DOS/VSE FORTRAN G Compiler Data Structures", IBM Press, New York, 1969
- 18 Johnson, Susanna., "DRM-210 Operating System Design". Nuclear Research Corporation internal report, 1990.
- 19 Kauler, Barry; "Windows Assembly Language and Systems Programming", CMP Books, Lawrence, KS, 1997
- 20 Kernighan, Brian W. and Ritchie, Dennis M., "The C Programming Language", Prentice-Hall, Englewood Cliffs, NJ, 1978
- 21 Lada, Charles J., Alves, João and Lada, Elizabeth A., "Infrared extinction and the Structure of the IC 5146 Dark Cloud", ApJ 512:250-259, 1999 Feb 10.

- 22 Lada, Charles J., Muensch, August A., Haisch, Karl E., Lada, Elizabeth A.,
Alves, Joao F., Tollestrup, Eric V. and Willner, S. P., "Infrared L-Band
Observations of the Trapezium Cluster: A Census of Circumstellar Disks and
Candidate Protostars", *Astronomical Journal*, 120:3162-3176, 2000 Dec.
- 23 Leach, Robert, "Analog Board", San Diego State University, San Diego, CA.,
undated internal report
- 24 Leach, Robert, "CCD Controller User's Manual", San Diego State University,
San Diego, CA., undated internal report
- 25 Leach, Robert, "DSP Software, Revision 2.30", San Diego State University, San
Diego, CA., undated internal report
- 26 Leach, Robert, "Optical and Infrared Cameral Electronics User's Manual", San
Diego State University, San Diego, CA., undated internal report
- 27 Leach, Robert, "Power Control Board, Revision 3B", San Diego State University,
San Diego, CA., dtd 3 Jan 1997, internal report

- 28 Leach, Robert, "Timing Board, Revision 6B", San Diego State University, San Diego, CA., undated internal report
- 29 Leach, Robert, "Utility Board, Revision 4B", San Diego State University, San Diego, CA., undated internal report
- 30 Microsoft Corporation, "Microsoft MS-DOS Programmer's Reference", Microsoft Press, Redmond, WA., 1986
- 31 Microsoft Corporation, "Microsoft Foundation Class Library Reference", Microsoft Press, Redmond, WA., 1995
- 32 Microsoft Corporation, "Microsoft Visual C++ Language Reference", Microsoft Press, Redmond, WA., 1997
- 33 Microsoft Corporation, "Microsoft Visual C++ Run-Time Library Reference", Microsoft Press, Redmond, WA., 1997

- 34 Morrison, Michael, "MFC In 24 Hours", Sams Publishing, Indianapolis, IN, 1999

- 35 Morrison, Ralph, "Grounding and Shielding Techniques in Instrumentation, 2nd Ed.", John Wiley & Sons, New York, 1977

- 36 Motorola Corporation, "DSP 56002 24-bit Digital Signal Processor User's Manual", Motorola, Inc. Semiconductor Products Sector, DSP Division, Austin, TX, 1995

- 37 Motorola Corporation, "Motorola DSP Assembler Reference Manual", Motorola, Inc., Semiconductor Products Sector, DSP Division, Austin, TX., 1996

- 38 Motorola Corporation, "Motorola DSP Linker/Librarian Reference Manual", Motorola, Inc., Semiconductor Products Sector, DSP Division, Austin, TX, 1996

- 39 Musciano, Chuck and Kennedy, Bill, "HTML, The Definitive Guide", O'Reilley & Associates, Inc., Sebastopol, CA., 1996

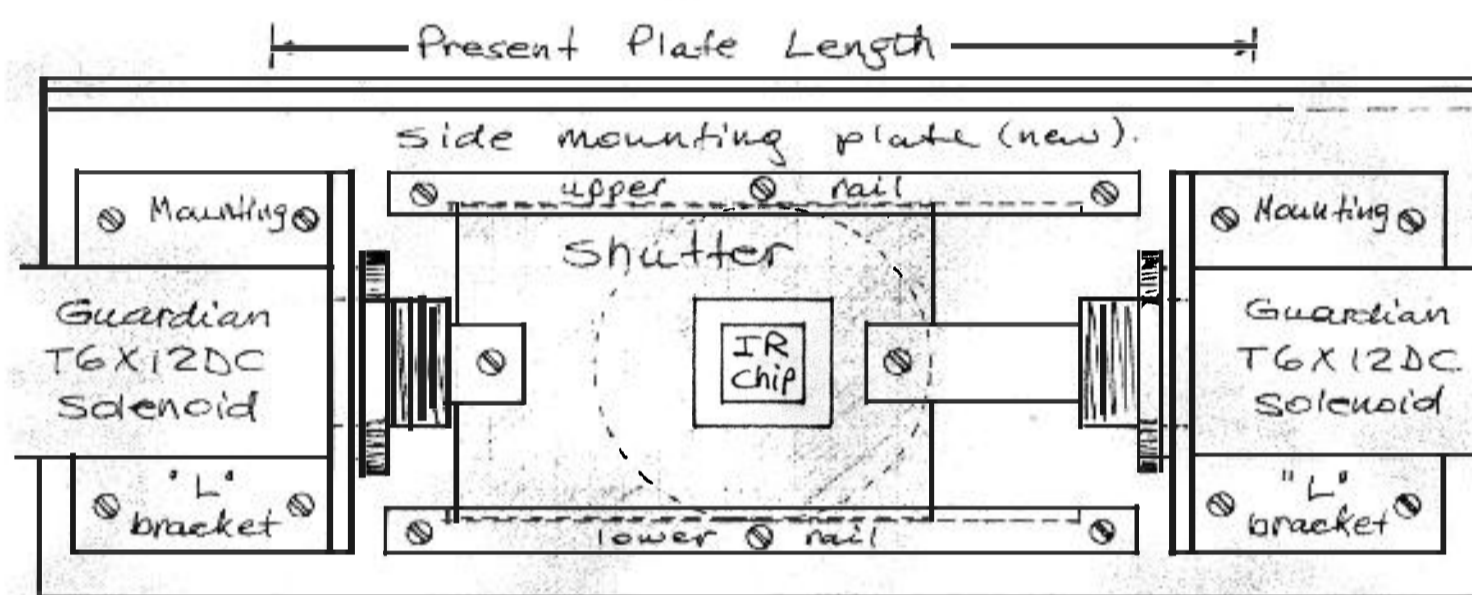
- 40 National Aeronautics and Space Administration, "Definition of the Flexible Image Transport System", NOST 100-2.0, NASA/Science Office of Standards and Technology, Greenbelt, MD, 1999
- 41 Neou, Vivian and Recker, Mimi, "HTML CD, An Internet Publishing Toolkit for Windows", Prentice Hall PTR, Upper Saddle River, NJ, 1996
- 42 National Instruments Corporation, "Lab Windows/CVI Master Index", National Instruments Corporation, Austin, TX., 1996
- 43 National Instruments Corporation, "Lab Windows/CVI Advanced Analysis Library Reference Manual", National Instruments Corporation, Austin, TX., 1996
- 44 National Instruments Corporation, "Getting Started with Lab Windows/CVI", National Instruments Corporation, Austin, TX, 1996
- 45 National Instruments Corporation, "Lab Windows/CVI Instrument Driver Developers Guide", National Instruments Corporation, Austin, TX., 1996

- 46 National Instruments Corporation, "Lab Windows/CVI Programmer Reference Manual", National Instruments Corporation, Austin, TX., 1996
- 47 National Instruments Corporation, "Lab Windows/CVI Standard Libraries Reference Manual", National Instruments Corporation, Austin, TX, 1996
- 48 National Instruments Corporation, "Lab Windows/CVI User Interface Reference Manual", National Instruments Corporation, Austin, TX., 1996
- 49 National Instruments Corporation, "Lab Windows/CVI User Manual", National Instruments Corporation, Austin, TX., 1996
- 50 Oppenheim, Alan V. and Schafer, Ronald W. "Discrete-Time Signal Processing", Prentice-Hall, Englewood Cliffs, NJ, 1989
- 51 Ott, Ellis R. "Process Quality Control: Troubleshooting and Interpretation of Data", McGraw-Hill, New York, 1990
- 52 Petzold, Charles, "Programming Windows 95", Microsoft Press, Redmond, WA., 1996

- 53 Rockwell International, untitled report on IR Labs P.O. 7597, reference number SC87004.FR, unpublished internal report dtd 1987
- 54 Rockwell International, "The CRL 128 x 128 Pixel Infrared Camera", internal report dtd 1 Jun 1988
- 55 Spectral Instruments, Inc., "PDCI Camera Interface Manual", internal report dtd 1999
- 56 Spectron Microsystems, "SpoX Internal Design", Spectron Microsystems, Santa Barbara, CA., 1993
- 57 Stone, Harold S., "Introduction to Computer Organization and Data Structures", McGraw-Hill, Inc., New York, 1972
- 58 Texas Instruments, "TMS320C6000 TCP/IP Stack Library Architectural Overview", Texas Instruments, Austin, TX, 1999

- 59 Walker, H. J., Heinrichsen, L., Richards, P. J., Klass, U. and Rasmussen, L. L.:
“ISOPHOT Observations of R CrB: A Star Caught Smoking”, *Astron Astrophys*,
315: L249-L252 (1996)
- 60 Wilkes, M. V., “Time-Sharing Computer Systems”, American Elsevier Publishing
Company, New York, 1968

**APPENDIX A. SCHEMATICS FOR RAO DESIGNED
ELECTRONICS PACKAGE**



(Twice the scale).

Infrared Detector Shutter and Mounting Plate.

(01-07-10).

Figure A-1. Infrared Detector Shutter and Mounting Plate (courtesy F. Babott)

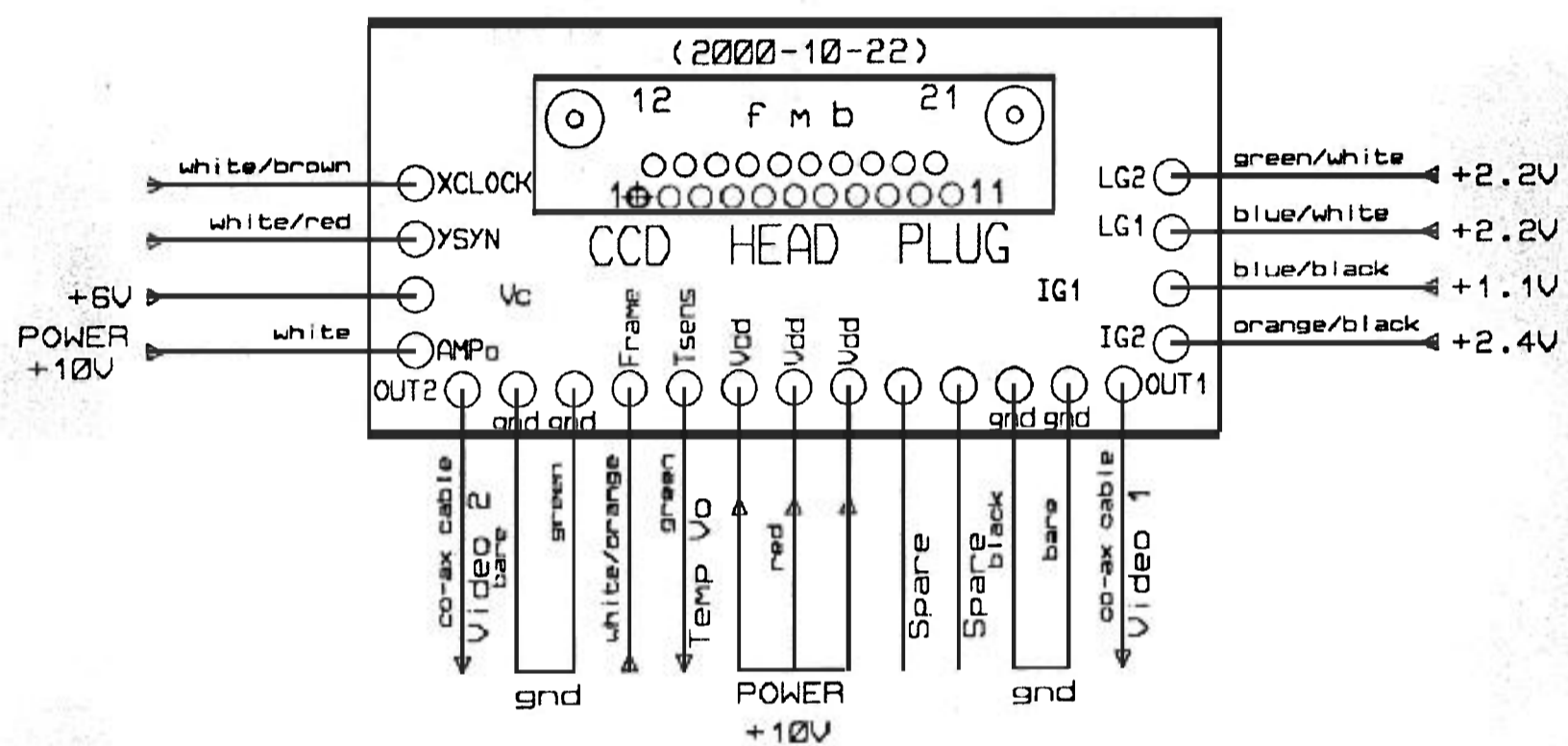


Figure A-2. Wiring Harness Card for Dewar Electronics (courtesy F. Babott)

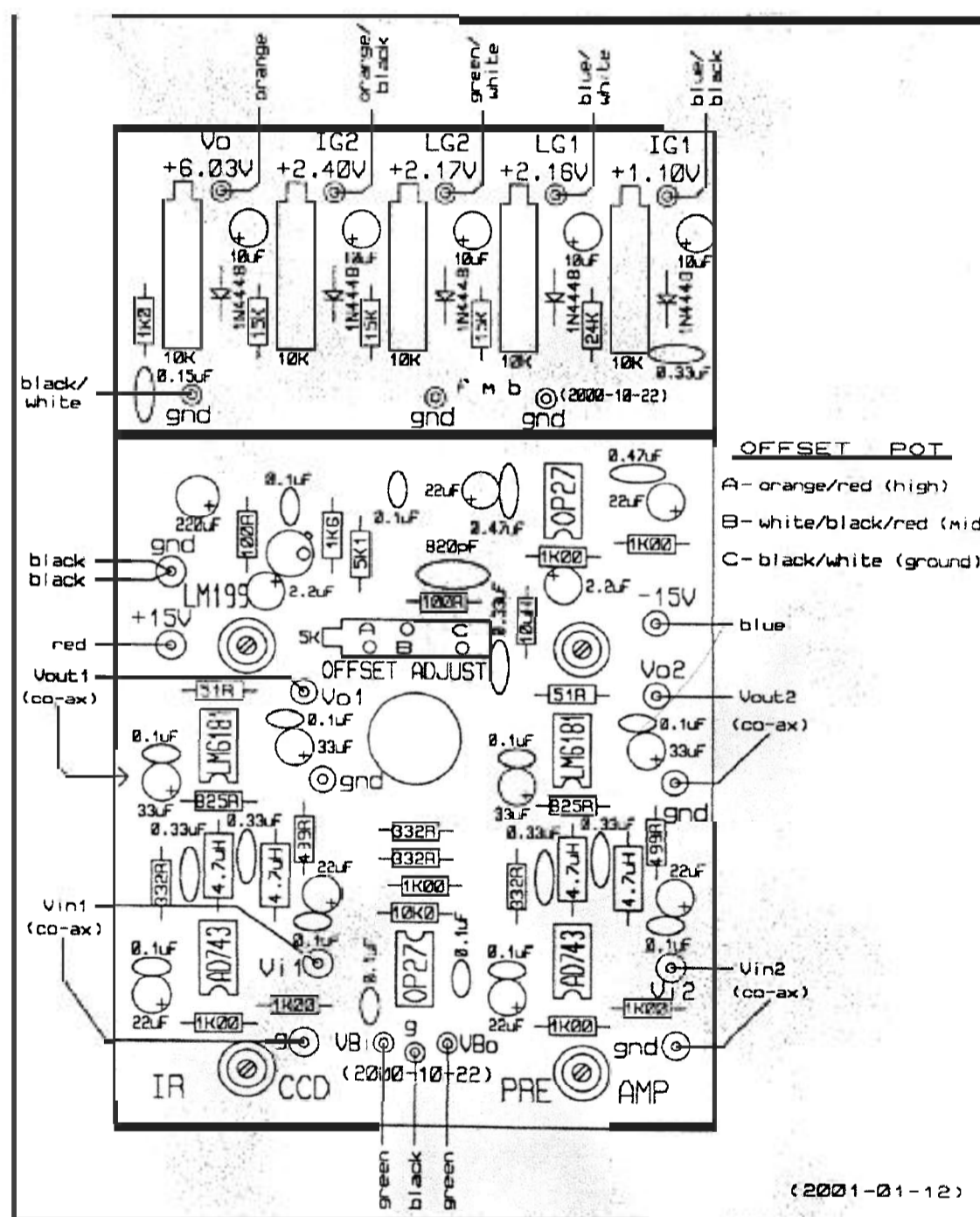


Figure A-3. Preamplifier and Bias Voltage Board Layouts (courtesy F. Babott)

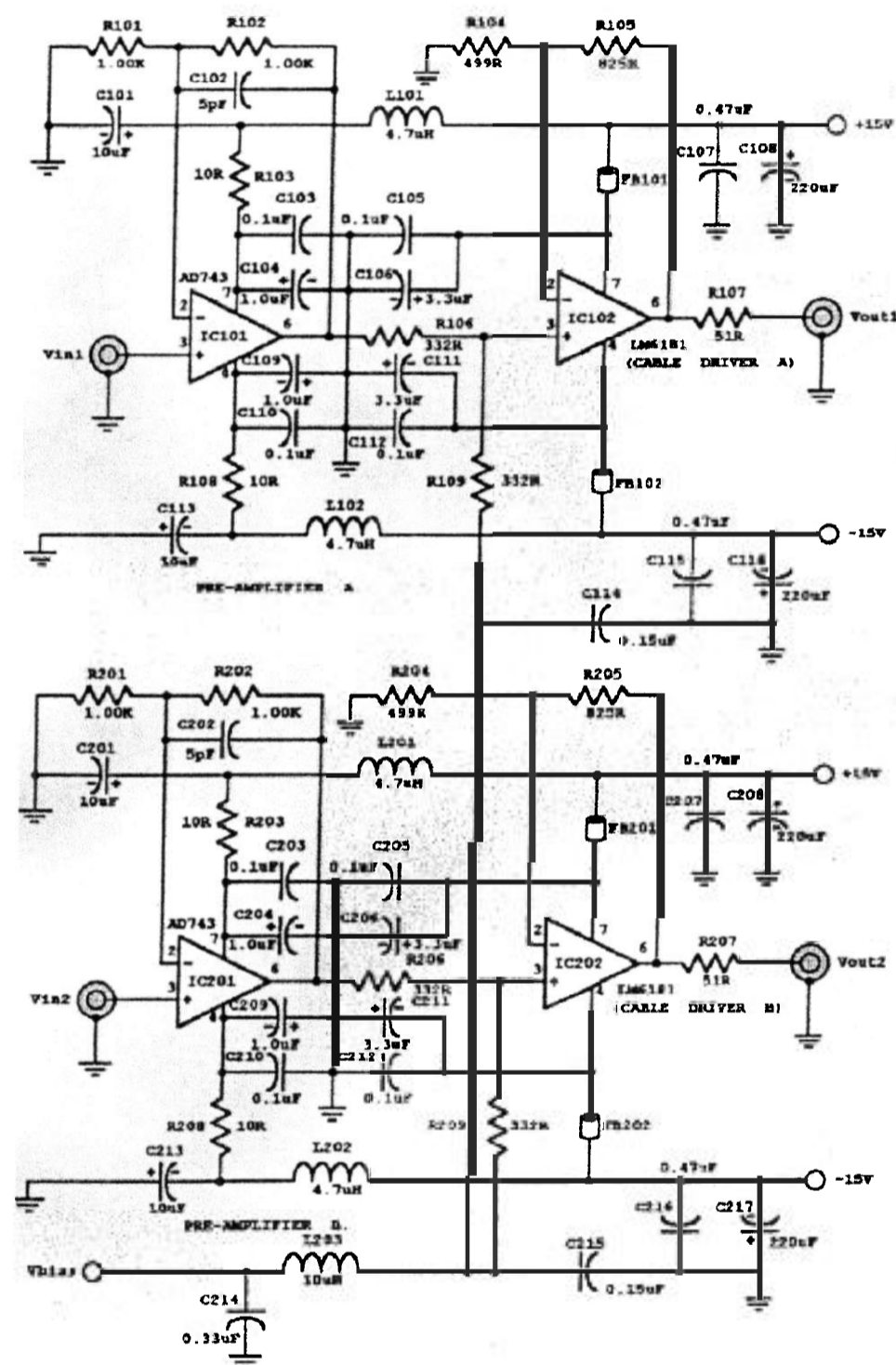


Figure A-4. Preamplifier Schematic (courtesy F. Babott)

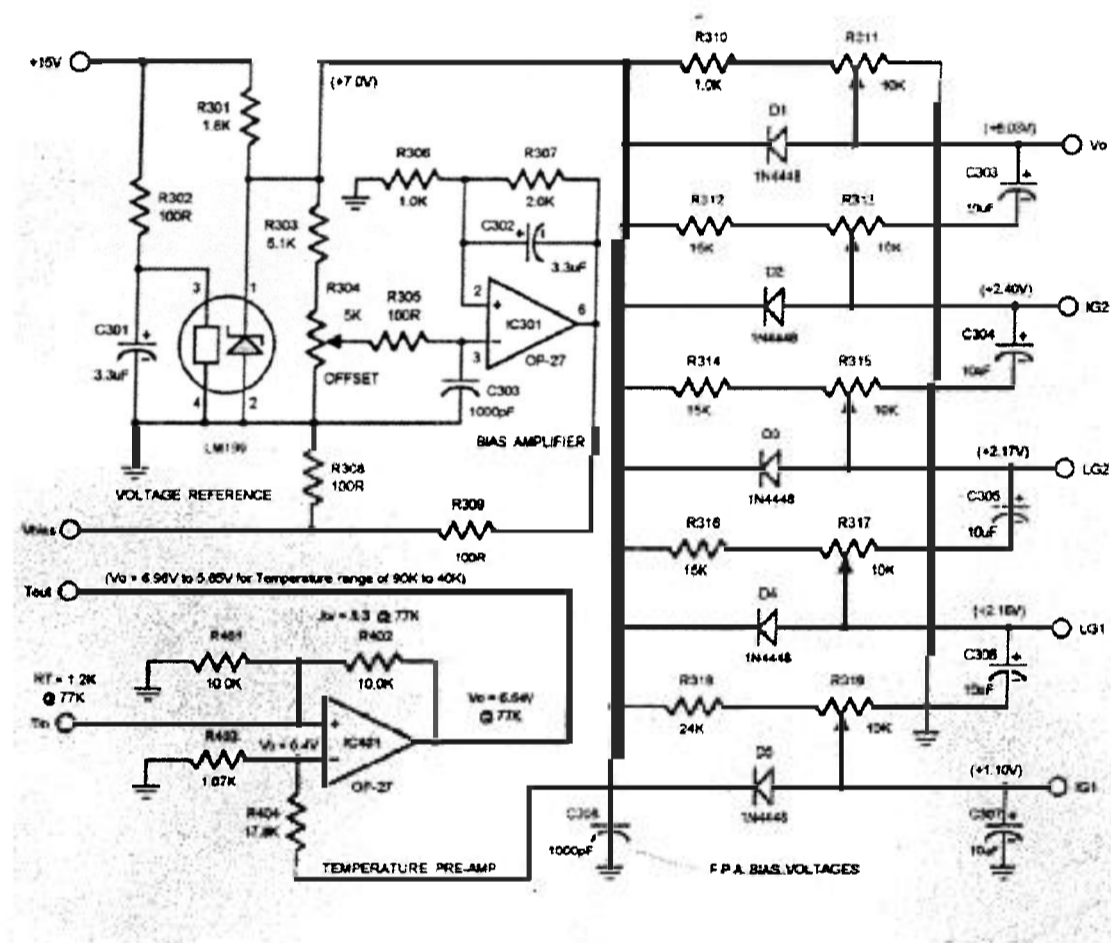


Figure A-5. Bias Voltage Generator Schematic (courtesy F. Babott)

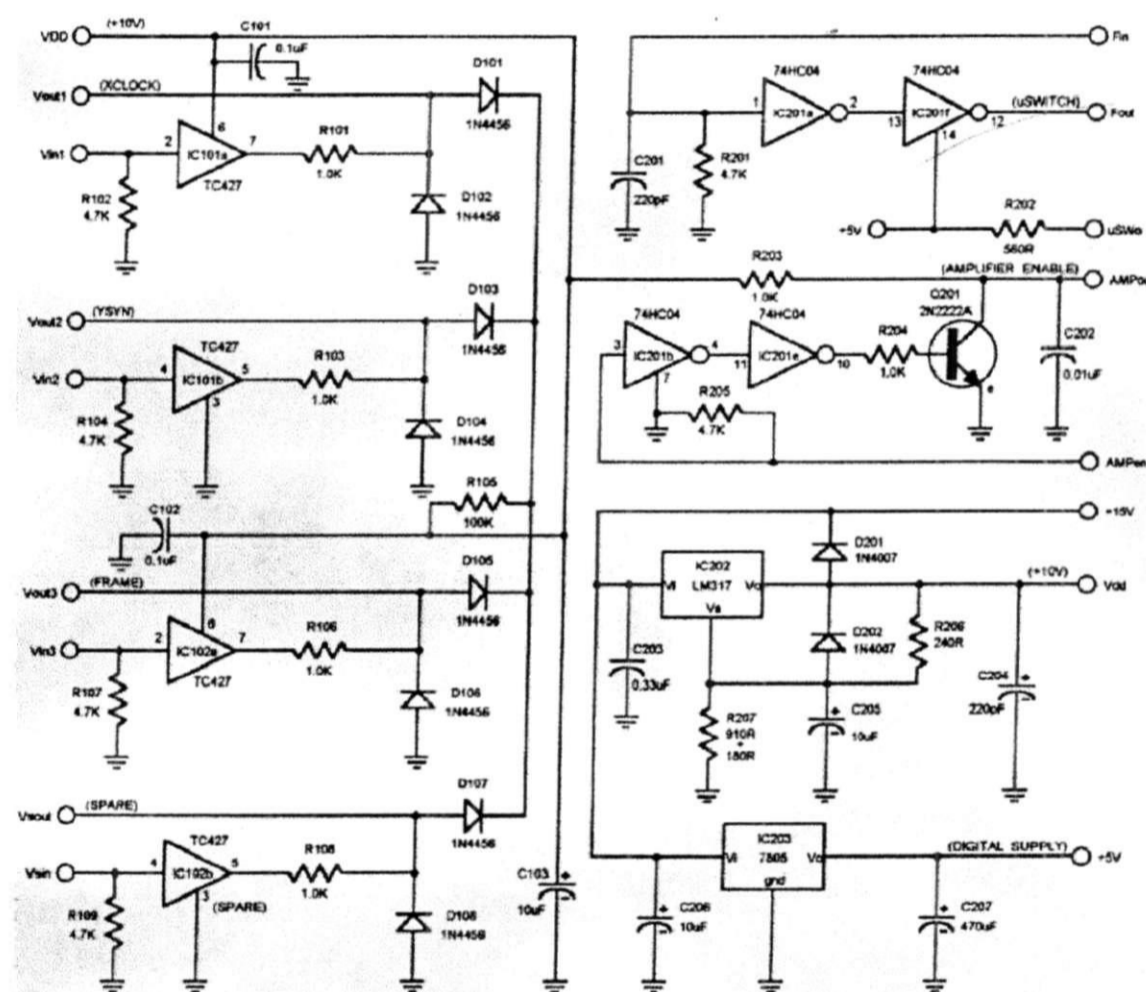


Figure A-7. Logic Board Schematic (partial) (courtesy F. Babott)

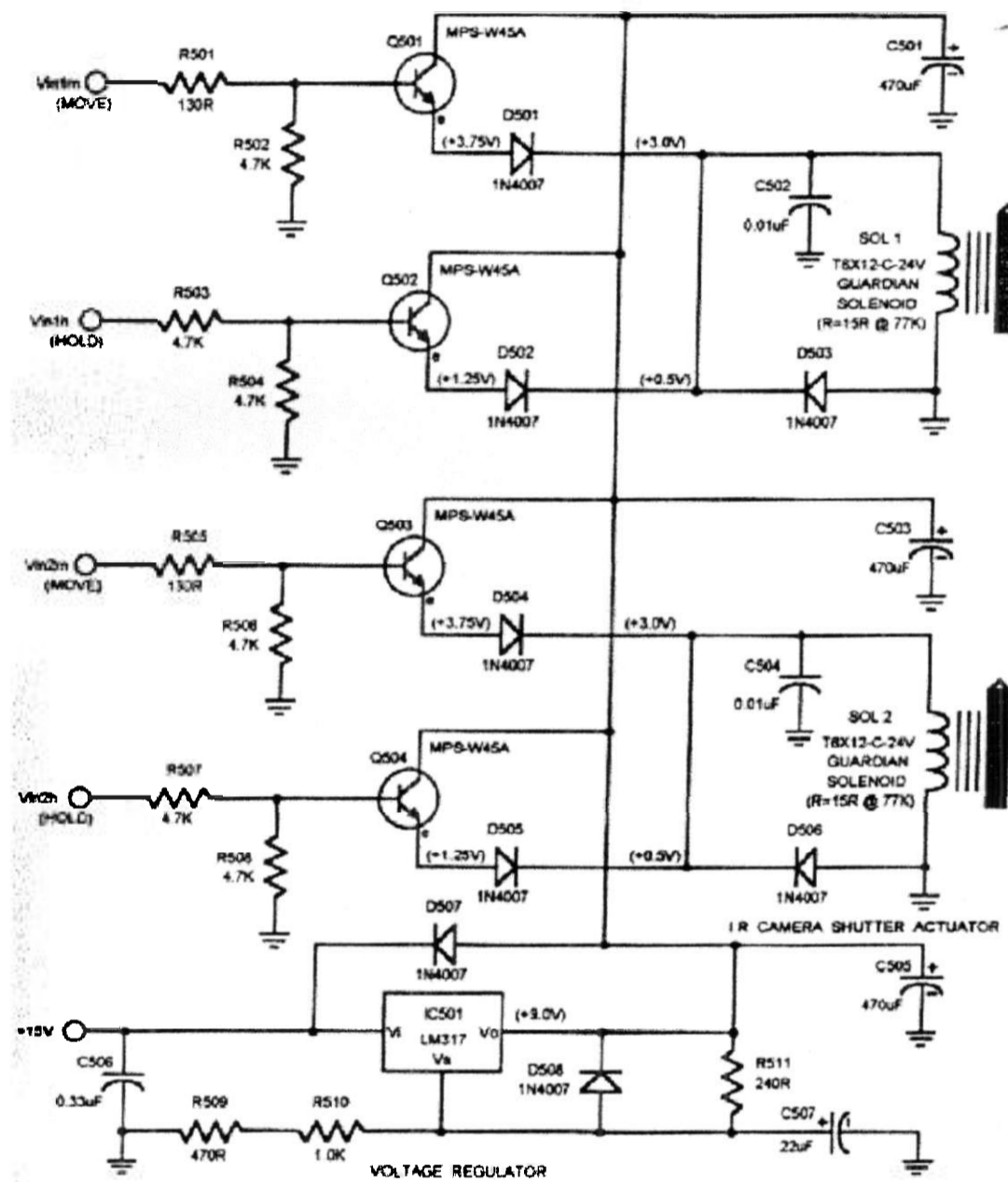


Figure A-8. Shutter Solenoid Driver Board Schematic (courtesy F. Babott)

**APPENDIX B. DEWAR CONNECTOR PINOUTS AND
IR LABS CONTROLLER PINOUT**

Table B-1.External Preamplifier Pinout

Pin	Type	Colour	Signal
A	Logic	WT/BK	GND
B	Logic	WT/BN	LS1
C	Logic	WT/RD	LS2
D	Logic	WT/OR	LS3
E	Logic	WT/YL	LS4
F	Spare		
G	Spare		
H	Spare		
I	Spare		
J	Spare		
K	Spare		
L	Solenoid	WT/BN	DVR1
M	Solenoid	WT/RD	DVR2
N	Solenoid	WT/OR	DVR3
P	Solenoid	WT/YL	DVR4
R	Spare		
S	Solenoid	WT/BK	GND
T	Logic	PU	Filter switch
U	Spare		
V	Spare		
W	Spare		
X	Logic	WT	Amplifier enable
Y	Bias	RD	+15
Z	Bias	BK/WT	GND
a	Bias	BK	GND
b	Bias	BU	-15

Table B-2. Preamplifier to Dewar Pinout

Pin	Type	Colour	Signal
A	Logic	GN	GND
B	Logic	GY	FI (Microswitch)
C	Logic	BU	Microswitch
D	Temp	GN	VB1 (TSENS)
E	Temp	BK	GND
F	Spare		
G	Logic	GN	GND
H	Spare		
J	Logic	WT	AMP Out (switched +10)
K	Bias	OR	VDD (+10)
L	Spare		
M	Video	coax	VID2
N	Video	coax	GND
P	Video	coax	VID1
R	Spare		
S	Bias	BK	GND
U	Bias	OR	V0
V	Bias	OR/BK	IG2
W	Bias	GN	LG2
X	Bias	BU/WT	LG1
Y	Bias	BU/BK	IG1
Z	Bias	BK	GND
a	Logic	BK	GND
b	Logic	BN	LS1 (XCLK)
c	Logic	RD	LS2 (YSYNC)
d	Logic	OR	LS3 (FRAME)
e	Logic	YL	LS4
f	Solenoid	GY	GND
g	Solenoid	BU	4 & 3
h	Solenoid	PU	GND
j	Solenoid	GN	1 & 2

Table B-3. Clock Generator Board DB-37 Connector

Pin	Designator	Function
1	CLK0	CLK
2	CLK1	YSYN
3	CLK2	FRAME
4	CLK3	Preamplifier Power
5	CLK4	Close shutter throw
6	CLK5	Close shutter hold
7	CLK6	Open shutter throw
8	CLK7	Open shutter hold
9	CLK8	Not populated
10	CLK9	Not populated
11	CLK10	Not populated
12	CLK11	Not populated
13	CLK12	Spare
14	CLK13	Spare
15	CLK14	Spare
16	CLK15	Spare
17	CLK16	Not populated
18	CLK17	Not populated
19	CLK18	Not populated
20	+12 Volts	Rail
21	-12 Volts	Rail
22	GND	
23	GND	
24	GND	
25	GND	
26	GND	
27	GND	
28	GND	
29	GND	
30	GND	
31	GND	
32	GND	
33	CLK19	Not populated
34	CLK20	Not populated
35	CLK21	Not populated
36	CLK22	Not populated
37	CLK23	Not populated

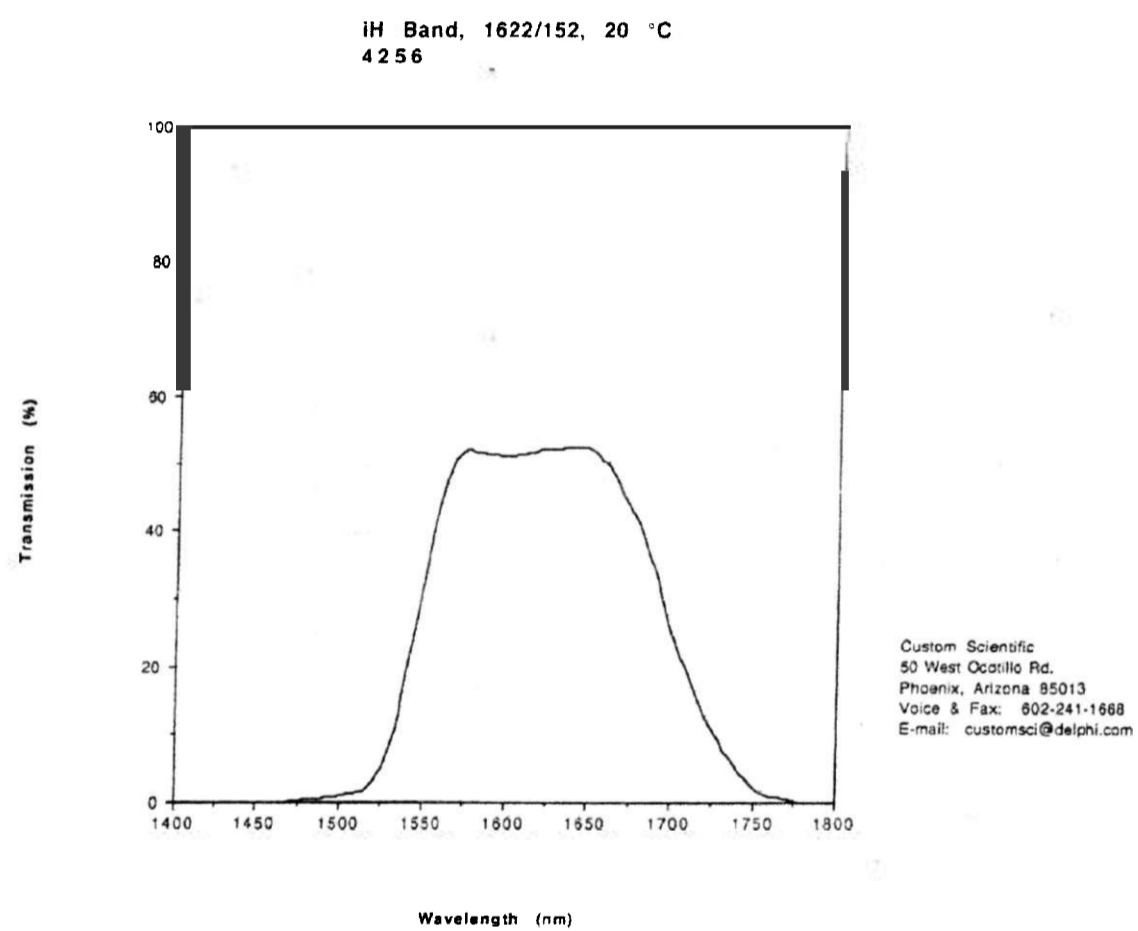
Table B-4. DB-37 for Controller to External Preamplifier Pinout

Pin	Designator	Function
1	CLK	Array clock
2	YSYN	Row Sync
3	FRAME	Frame
4	POWER	Amp/chip enable
5	CSH	close shutter
6	CSHH	close shutter hold
7	OSH	open shutter
8	OSHH	open shutter hold
9	spare	
10	spare	
11	+15VDC	
12	+15VDC	
13	-15VDC	
14	-15VDC	
15	GND	
16	GND	
17	spare	
18	spare	
19	spare	
20	+15VDC	
21	-15VDC	
22	GND	
23	GND	
24	GND	
25	GND	
26	GND	
27	GND	
28	GND	
29	GND	
30	GND	
31	GND	
32	GND	
33	spare	
34	spare	
35	amp power on	
36	spare	
37	microswitch	

Table B-5. DB-25 for Controller to Filter Wheel Controller Pinout

Pin	Designator	Function
1	+15VDC	DC PWR
2	+15VDC	DC PWR
3	unused	do not use
4	-15VDC	DC PWR
5	-15VDC	DC PWR
6	unused	do not use
7	GND	DC PWR
8	GND	DC PWR
9	spare	
10	spare	
11	spare	
12	spare	
13	spare	
14	spare	
15	spare	
16	spare	
17	spare	
18	spare	
19	spare	
20	spare	
21	spare	
22	amp power on	to filter cntl
23	spare	
24	gnd	to filter cntl
25	microswitch	to filter cntl

APPENDIX C. NARROW BAND FILTER CHARACTERISTICS

Figure C-1. iH Filter Passband¹

Data for iH, iK, and iL filter passbands courtesy Custom Scientific, 50 West Ocotillo Road, Phoenix, AZ, 85013. Voice and fax, 602-241-1668. E-mail, customsci@delphi.com.

Ik Band
2195, 188
4153

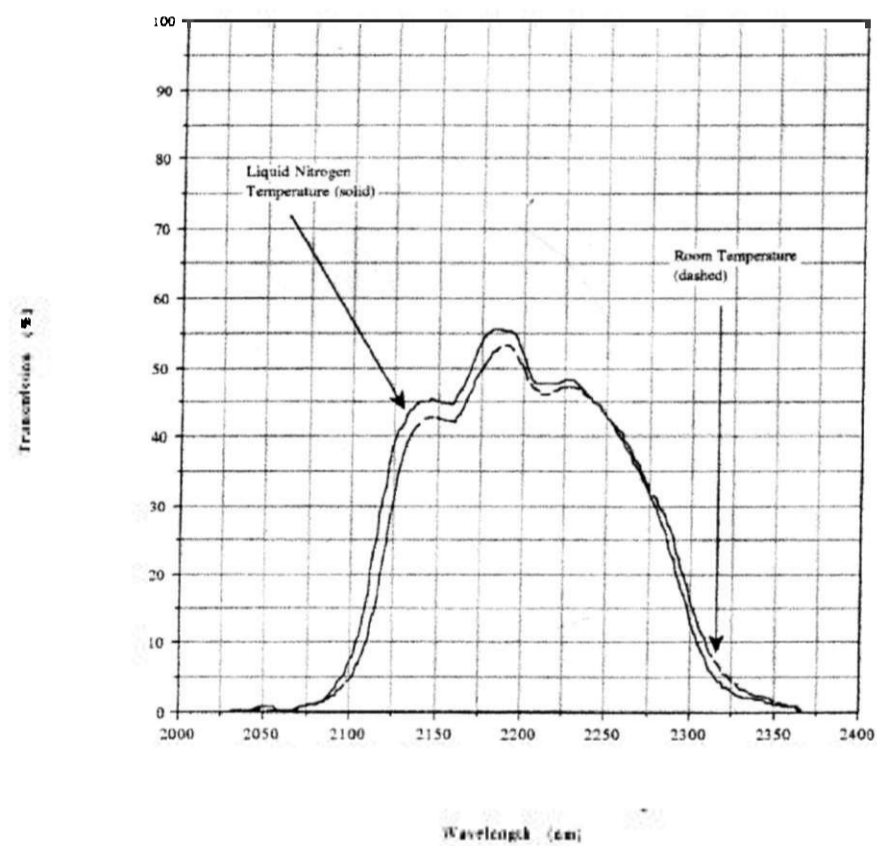


Figure C-2. iK Filter Passband

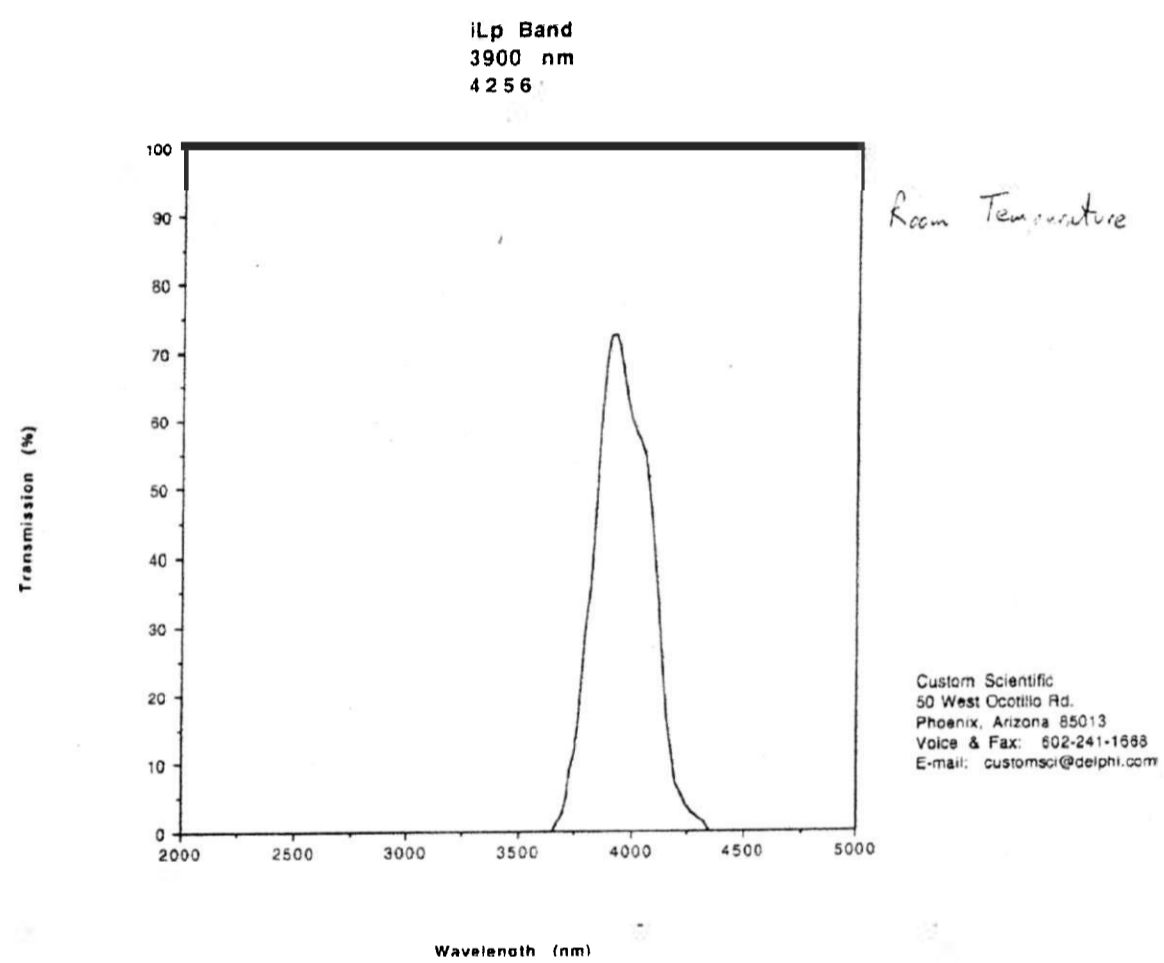


Figure C-3. iL Filter Passband

Comments: Filter scanned at 93 K and normal to the light path.

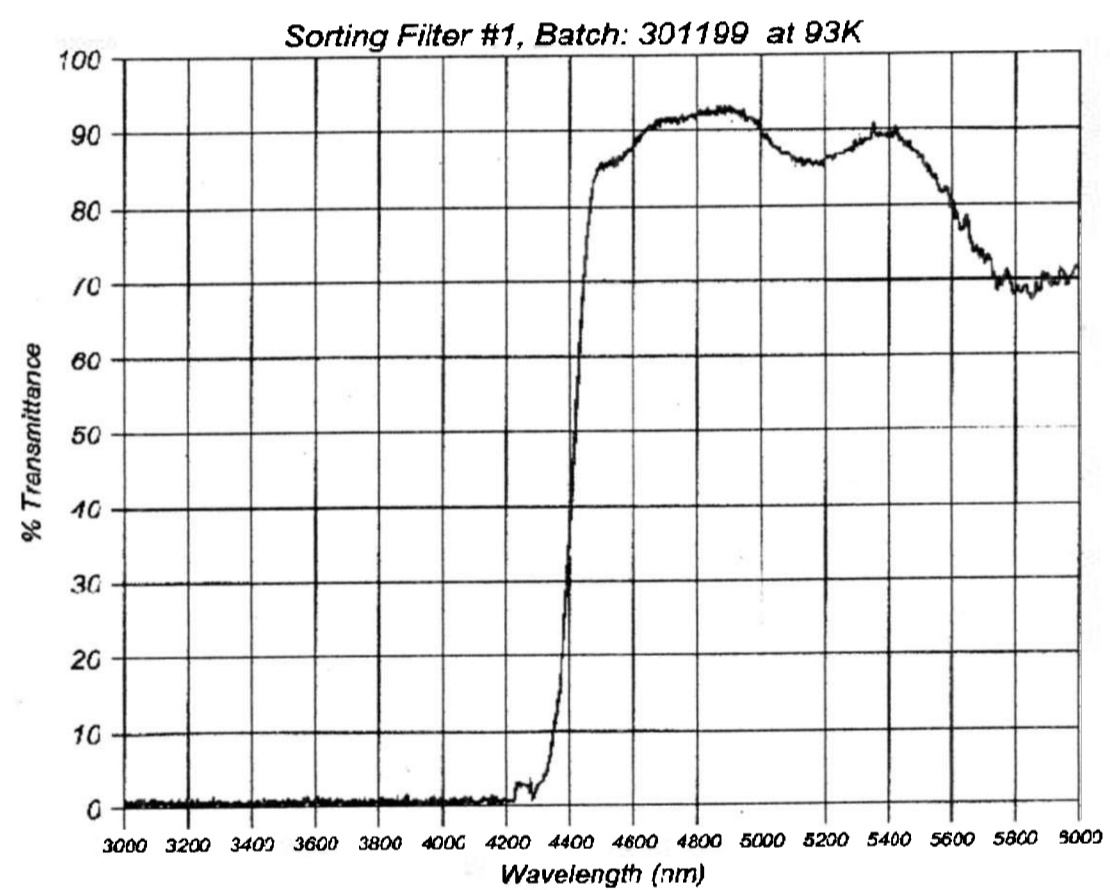


Figure C-4. Standard M Band Filter Passband²

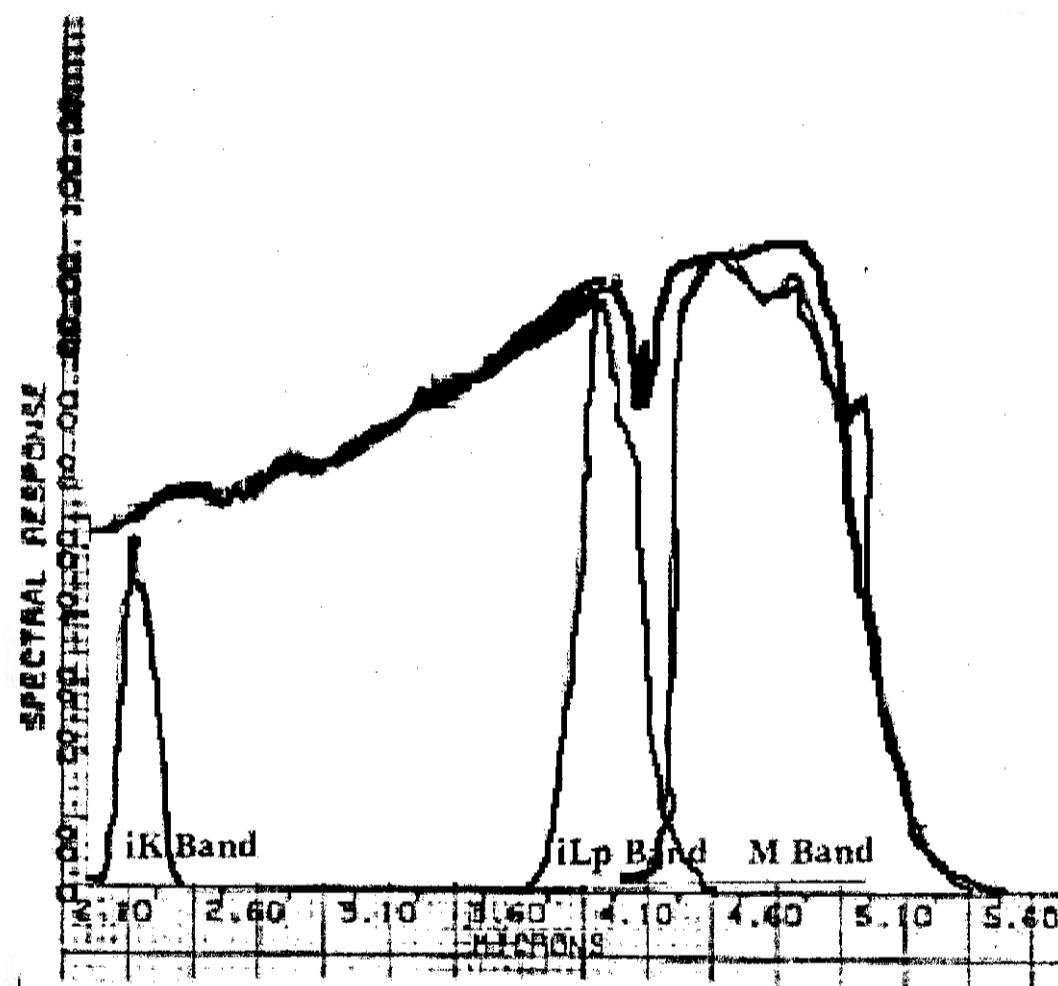


Figure C-5. TCM-1000C Response Curve with
iK, iLp, and M Filter Passbands Overlaid³

³

iH filter passband is not overlaid because it is out of the range for this detector.

Table C-1. Central Wavelengths and Zero Magnitude Fluxes at
Different Photometric Bands⁴.

Filter	λ_o (μm)	F_{vo} (Jy)
U	0.365	1720
B	0.440	4490
V	0.55	3360
R	0.70	2780
I	0.90	2240
J	1.25	1580
H	1.65	980
K	2.20	632
L	3.8	274
M	4.8	156
N	10.5	38.4
Q	20	9.8
Z	25	6.5

⁴
Kwok, Sun, Op. Cit.

APPENDIX D. SOURCE CODE LISTINGS

D.1.1. List of Program Files in Host Program

SIW95DLL.dll
SIW95DLL.lib
SIW95VXD.vxd
SIW95DLL.h
host.c
GLOBAL.H
host.prj
dllmak.prj
main.uir
main.h
debugmenu.uir
debugmenu.h
safety.h
xlate.h
xlate.c
filemenu.h
imagemenu.h
filemenu.uir
FITSLIB.lib
imagemenu.uir
iob.h
errmenu.h
cfitsio.def
errmenu.uir
fitsio.h
longnam.h
setupmenu.h
filtermenu.h
setupmenu.uir
filtermenu.uir
msctype.c
msctype.h
iob.h.bak
ACCES32.h
startup.h
ACCES32.DLL
CBACCES.lib
startup.uir

indian.raw
mod-testbw1.jpg
host.ico
ircamera.exe
IR Camera.lnk
IR CAMERA.ico
wrapper.dll
config.dat
fitslib.h
mod-test800.gif
Indian_Head_320.jpg
tcf_sydney.jpg
sydney.raw
wtindian.raw
about.h
about.uir
svid.h

D.1.2 Listing of host.Prj

```
[Project Header]
Version = 501
Platform Code = 4
Pathname = "/f/milone/host/host.prj"
CVI Dir = "/d/cvi"
VXIplug&play Framework Dir = "/C/VXIPNP/win95"
Number of Files = 18
Sort Type = "No Sort"
Target Type = "Executable"
Flags = 16
Drag Bar Left = 165
Window Top = 169
Window Left = 194
Window Bottom = 569
Window Right = 854

[File 0001]
File Type = "Unknown"
Path = "/f/milone/host/IR CAMERA.ico"
Res Id = 1
Exclude = False
Disk Date = 3075329814
Project Flags = 0
Window Top = 0
Window Left = 0
Window Height = 0
Window Width = 0
Source Window State = "0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,"

[File 0002]
File Type = "Unknown"
Path = "/f/milone/host/IR Camera.lnk"
Res Id = 2
Exclude = False
Disk Date = 3075331662
Project Flags = 0
Window Top = 0
Window Left = 0
Window Height = 0
```

Window Width = 0
 Source Window State = "0,"

[File 0003]
 File Type = "Include"
 Path = "/f/milone/host/SIW95DLL.h"
 Res Id = 3
 Exclude = False
 Disk Date = 3072967304
 Project Flags = 0
 Window Top = 140
 Window Left = 66
 Window Height = 0
 Window Width = 0
 Source Window State = "1,0,0,0,0,0,0,0,0,80,0,0,0,0,0,25,0,0,6,31,"

[File 0004]
 File Type = "Library"
 Path = "/f/milone/host/SIW95DLL.lib"
 Res Id = 4
 Exclude = False
 Disk Date = 3072967760
 Project Flags = 0
 Window Top = 0
 Window Left = 0
 Window Height = 0
 Window Width = 0

[File 0005]
 File Type = "CSource"
 Path = "/f/milone/host/host.c"
 Res Id = 5
 Exclude = False
 Disk Date = 3079112322
 Project Flags = 0
 Window Top = 23
 Window Left = 0
 Window Height = 0
 Window Width = 0
 Source Window State = "1,1392,1397,1392,0,-1,0,0,0,124,0,1,0,1,0,46,920,0,931,12,"
 Header Dependencies Line0001 =
 "1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30"
 Header Dependencies Line0002 = ",31,32,33,34,35,36,37,38,"

[File 0006]

File Type = "Library"

Path = "/f/milone/host/CBACCES.lib"

Res Id = 6

Exclude = False

Disk Date = 3023254800

Project Flags = 0

Window Top = 0

Window Left = 0

Window Height = 0

Window Width = 0

[File 0007]

File Type = "CSource"

Path = "/f/milone/host/xlate.c"

Res Id = 7

Exclude = False

Disk Date = 3075244476

Project Flags = 0

Window Top = 85

Window Left = 5

Window Height = 0

Window Width = 0

Source Window State = "1,2,2,2,0,15,16,0,0,80,0,1,0,1,0,25,49,0,61,7,"

Header Dependencies = "1,2,3,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,39,"

[File 0008]

File Type = "CSource"

Path = "/f/milone/host/msctype.c"

Res Id = 8

Exclude = False

Disk Date = 3075237850

Project Flags = 0

Window Top = 23

Window Left = 0

Window Height = 0

Window Width = 0

Source Window State = "1,14,14,14,0,14,15,0,0,126,0,0,0,0,0,38,0,0,13,26,"

Header Dependencies = "39,"

[File 0009]

File Type = "Library"

Path = "/f/milone/host/FITSLIB.lib"

Res Id = 9
Exclude = False
Disk Date = 3075467186
Project Flags = 0
Window Top = 0
Window Left = 0
Window Height = 0
Window Width = 0

[File 0010]
File Type = "User Interface Resource"
Path = "/f/milone/host/errmenu.uir"
Res Id = 10
Exclude = False
Disk Date = 3075290604
Project Flags = 0
Window Top = 209
Window Left = 102
Window Height = 400
Window Width = 660

[File 0011]
File Type = "User Interface Resource"
Path = "/f/milone/host/debugmenu.uir"
Res Id = 11
Exclude = False
Disk Date = 3079026060
Project Flags = 0
Window Top = 94
Window Left = 42
Window Height = 400
Window Width = 676

[File 0012]
File Type = "User Interface Resource"
Path = "/f/milone/host/filemenu.uir"
Res Id = 12
Exclude = False
Disk Date = 3075306526
Project Flags = 0
Window Top = 117
Window Left = 54
Window Height = 400

Window Width = 660

[File 0013]

File Type = "User Interface Resource"

Path = "/f/milone/host/filtermenu.uir"

Res Id = 13

Exclude = False

Disk Date = 3076010050

Project Flags = 0

Window Top = 94

Window Left = 42

Window Height = 400

Window Width = 660

[File 0014]

File Type = "User Interface Resource"

Path = "/f/milone/host/imagemenu.uir"

Res Id = 14

Exclude = False

Disk Date = 3079033344

Project Flags = 0

Window Top = 94

Window Left = 42

Window Height = 400

Window Width = 660

[File 0015]

File Type = "User Interface Resource"

Path = "/f/milone/host/main.uir"

Res Id = 15

Exclude = False

Disk Date = 3079018116

Project Flags = 0

Window Top = 94

Window Left = 42

Window Height = 400

Window Width = 660

[File 0016]

File Type = "User Interface Resource"

Path = "/f/milone/host/setupmenu.uir"

Res Id = 16

Exclude = False

Disk Date = 3077128012
Project Flags = 0
Window Top = 163
Window Left = 78
Window Height = 400
Window Width = 660

[File 0017]
File Type = "User Interface Resource"
Path = "/f/milone/host/startup.uir"
Res Id = 17
Exclude = False
Disk Date = 3079097350
Project Flags = 0
Window Top = 48
Window Left = 18
Window Height = 400
Window Width = 660

[File 0018]
File Type = "User Interface Resource"
Path = "/f/milone/host/about.uir"
Res Id = 18
Exclude = False
Disk Date = 3079035078
Project Flags = 0
Window Top = 278
Window Left = 138
Window Height = 400
Window Width = 660

[Compiler Options]
Default Calling Convention = "cdecl"
Max Number Of Errors = 100
Require Prototypes = True
Require Return Values = True
Enable Pointer Mismatch Warning = False
Enable Unreachable Code Warning = False
Track Include File Dependencies = True
Prompt For Missing Includes = True
Stop On First Error File = False
Bring Up Err Win For Warnings = True
Show Build Dialog = True

[Run Options]

Stack Size = 250000
Debugging Level = "None"
Save Changes Before Running = "Ask"
Break On Library Errors = True
Hide Windows = False
Unload DLLs After Each Run = True
Check Disk Dates Before Each Run = True
Break At First Statement = False

[Build Options]

DLL Debugging Level = "None"

[Compiler Defines]

Compiler Defines = "/DWIN32_LEAN_AND_MEAN"

[Command Line Args]

Command Line Args = ""

[Included Headers]

Header 0026 = "/f/milone/host/SIW95DLL.h"
Header 0001 = "/d/cvi/include/utility.h"
Header 0002 = "/d/cvi/include/cvdef.h"
Header 0003 = "/d/cvi/include/cvirte.h"
Header 0004 = "/d/cvi/include/rs232.h"
Header 0005 = "/d/cvi/include/ansi_c.h"
Header 0006 = "/d/cvi/include/ansi/assert.h"
Header 0007 = "/d/cvi/include/ansi/ctype.h"
Header 0008 = "/d/cvi/include/ansi/errno.h"
Header 0009 = "/d/cvi/include/ansi/float.h"
Header 0010 = "/d/cvi/include/ansi/limits.h"
Header 0011 = "/d/cvi/include/ansi/locale.h"
Header 0012 = "/d/cvi/include/ansi/math.h"
Header 0013 = "/d/cvi/include/ansi/setjmp.h"
Header 0014 = "/d/cvi/include/ansi/signal.h"
Header 0015 = "/d/cvi/include/ansi/stdarg.h"
Header 0016 = "/d/cvi/include/ansi/stddef.h"
Header 0017 = "/d/cvi/include/ansi/stdio.h"
Header 0018 = "/d/cvi/include/ansi/stdlib.h"
Header 0019 = "/d/cvi/include/ansi/string.h"
Header 0020 = "/d/cvi/include/ansi/time.h"
Header 0021 = "/d/cvi/include/formatio.h"
Header 0022 = "/d/cvi/include/userint.h"

Header 0023 = "/f/milone/host/GLOBAL.H"
 Header 0024 = "/f/milone/host/svid.h"
 Header 0025 = "/f/milone/host/xlate.h"
 Header 0027 = "/f/milone/host/fitslib.h"
 Header 0028 = "/f/milone/host/longnam.h"
 Header 0029 = "/f/milone/host/ACCES32.h"
 Header 0030 = "/f/milone/host/startup.h"
 Header 0031 = "/f/milone/host/main.h"
 Header 0032 = "/f/milone/host/filemenu.h"
 Header 0033 = "/f/milone/host/debugmenu.h"
 Header 0034 = "/f/milone/host/errmenu.h"
 Header 0035 = "/f/milone/host/imagemenu.h"
 Header 0036 = "/f/milone/host/setupmenu.h"
 Header 0037 = "/f/milone/host/filtermenu.h"
 Header 0038 = "/f/milone/host/about.h"
 Header 0039 = "/f/milone/host/msctype.h"
 Max Header Number = 39

[Create Executable]

Executable File = "/f/milone/host/ircamera.exe"
 Icon File = "/f/milone/host/IR CAMERA.ico"
 Application Title = "IR CAMERA"
 Numeric File Version = "1,0,0,0"
 Numeric Prod Version = "1,0,0,0"
 Comments = ""
 Company Name = "UNIVERSITY OF CALGARY"
 File Version = "0.14 TEST"
 Legal Copyright = "Copyright © Quantum Magnetics, 2001"
 Legal Trademarks = "Copyright (c) 2001 UNIVERSITY OF CALGARY"
 Private Build = ""
 Product Version = "0.14 TEST"
 Special Build = ""
 DLL Exports = "Include File Symbols"
 DLL Import Library Choice = "Gen Lib For Current Mode"
 Use VXIPNP Subdirectories for Import Libraries = False
 Use Dflt Import Lib Base Name = True
 Where to Copy DLL = "Do not copy"
 Add Type Lib To DLL = False
 Include Type Lib Help Links = False
 Type Lib FP File = ""
 Type Lib Guid = ""
 Instrument Driver Support Only = False

[External Compiler Support]

Create UIR Callbacks File = False
 Using LoadExternalModule = False
 Create Project Symbols File = True
 UIR Callbacks Obj File = ""
 Project Symbols H File = ""
 Project Symbols Obj File = ""

[DLL Debugging Support]

External Process Path = ""

[DLLs Used By Executable]

DLL 0001 = "/f/milone/host/SIW95DLL.dll"
 DLL 0002 = "/f/milone/host/ACCES32.DLL"
 DLL 0003 = "/f/milone/host/wrapper.dll"

[Distribution Kit]

Installation Directory = "ircamera"
 Install Run-Time Engine = True
 Install Low-Level Support Driver = True
 Media Size = 3
 kBytes Reserved on First Disk = 0
 Target Path = "/f/milone/host/build"
 Language = "English"
 Core Group Index = 1
 DLL Group Index = 2
 Project File 0001 = "/f/milone/host/errmenu.uir"
 Project File 0002 = "/f/milone/host/debugmenu.uir"
 Project File 0003 = "/f/milone/host/filemenu.uir"
 Project File 0004 = "/f/milone/host/filtermenu.uir"
 Project File 0005 = "/f/milone/host/imagemenu.uir"
 Project File 0006 = "/f/milone/host/main.uir"
 Project File 0007 = "/f/milone/host/setupmenu.uir"
 Project File 0008 = "/f/milone/host/startup.uir"
 Project File 0009 = "/f/milone/host/about.uir"
 Project File 0010 = "/f/milone/host/ircamera.exe"
 Project File 0011 = "/f/milone/host/SIW95DLL.dll"
 Project File 0012 = "/f/milone/host/ACCES32.DLL"
 Project File 0013 = "/f/milone/host/wrapper.dll"
 Use Custom Script = False
 Run Executable After Setup Group Index = -1
 Run Executable After Setup File Index = -1
 Use Default Program Group Name = True

Use Default Installation Name = True

[Distribution Kit File Group 001]

Group Name = "IR CAMERA Files"

Destination Directory = "Application"

Use Relative Path = False

Install Icons = True

Distribute Multiple Objects = False

Replace Mode = "Ask"

File 0001 = "/f/milone/host/ircamera.exe"

[Distribution Kit File Group 002]

Group Name = "DLL Files"

Destination Directory = "Application"

Use Relative Path = False

Install Icons = False

Distribute Multiple Objects = False

Replace Mode = "Ask"

File 0001 = "/f/milone/host/ACCES32.DLL"

File 0002 = "/f/milone/host/SIW95DLL.dll"

File 0003 = "/f/milone/host/wrapper.dll"

[Distribution Kit File Group 003]

Group Name = "VXD FILES"

Destination Directory = "Windows System"

Use Relative Path = False

Install Icons = False

Distribute Multiple Objects = False

Replace Mode = "Ask"

File 0001 = "/f/milone/host/SIW95VXD.vxd"

[Distribution Kit File Group 004]

Group Name = "TEST IMAGES"

Destination Directory = "Application"

Use Relative Path = False

Install Icons = False

Distribute Multiple Objects = False

Replace Mode = "Ask"

File 0001 = "/f/milone/host/indian.raw"

File 0002 = "/f/milone/host/sydney.raw"

File 0003 = "/f/milone/host/wtindian.raw"

[Distribution Kit File Group 005]

Group Name = "LINK"
 Destination Directory = "Windows"
 Use Relative Path = True
 Relative Path = "desktop"
 Install Icons = False
 Distribute Multiple Objects = False
 Replace Mode = "Ask"
 File 0001 = "/f/milone/host/TR Camera.lnk"

[Distribution Kit File Group 006]
 Group Name = "HELPPFILES"
 Destination Directory = "Application"
 Use Relative Path = True
 Relative Path = "hlp"
 Install Icons = False
 Distribute Multiple Objects = False
 Replace Mode = "Ask"
 File 0001 = "/f/milone/host/hlp/directory.html"
 File 0002 = "/f/milone/host/hlp/help.html"
 File 0003 = "/f/milone/host/hlp/history.html"
 File 0004 = "/f/milone/host/hlp/index.html"
 File 0005 = "/f/milone/host/hlp/intro.html"
 File 0006 = "/f/milone/host/hlp/legal.html"
 File 0007 = "/f/milone/host/hlp/page1.html"

[Distribution Kit File Group 007]
 Group Name = "UIR"
 Destination Directory = "Application"
 Use Relative Path = False
 Install Icons = False
 Distribute Multiple Objects = False
 Replace Mode = "Ask"
 File 0001 = "/f/milone/host/about.uir"
 File 0002 = "/f/milone/host/debugmenu.uir"
 File 0003 = "/f/milone/host/errmenu.uir"
 File 0004 = "/f/milone/host/filemenu.uir"
 File 0005 = "/f/milone/host/filtermenu.uir"
 File 0006 = "/f/milone/host/imagemenu.uir"
 File 0007 = "/f/milone/host/main.uir"
 File 0008 = "/f/milone/host/setupmenu.uir"
 File 0009 = "/f/milone/host/startup.uir"

D.1.3 Listing of dllmak.prj

```

[Project Header]
Version = 501
Platform Code = 4
Pathname = "/f/milone/host/dllmak.prj"
CVI Dir = "/d/cvi"
VXIplug&play Framework Dir = "/C/VXIPNP/win95"
Number of Files = 1
Sort Type = "No Sort"
Target Type = "Dynamic Link Library"
Flags = 16
Drag Bar Left = 165
Window Top = 24
Window Left = 5
Window Bottom = 424
Window Right = 665

[File 0001]
File Type = "Include"
Path = "/f/milone/host/SIW95DLL.h"
Res Id = 1
Exclude = False
Disk Date = 3072967304
Project Flags = 0
Window Top = 94
Window Left = 42
Window Height = 0
Window Width = 0
Source Window State = "1,0,0,0,0,0,0,0,80,0,0,0,0,25,0,0,0,0,"

[Compiler Options]
Default Calling Convention = "cdecl"
Max Number Of Errors = 100
Require Prototypes = True
Require Return Values = True
Enable Pointer Mismatch Warning = True
Enable Unreachable Code Warning = True
Track Include File Dependencies = True
Prompt For Missing Includes = True
Stop On First Error File = False

```

Bring Up Err Win For Warnings = True
Show Build Dialog = True

[Run Options]
Stack Size = 250000
Debugging Level = "None"
Save Changes Before Running = "Ask"
Break On Library Errors = True
Hide Windows = False
Unload DLLs After Each Run = True
Check Disk Dates Before Each Run = True
Break At First Statement = False

[Build Options]
DLL Debugging Level = "None"

[Compiler Defines]
Compiler Defines = "/DWIN32_LEAN_AND_MEAN"

[Command Line Args]
Command Line Args = ""

[Included Headers]
Max Header Number = 0

[Create Executable]
Executable File = ""
Icon File = ""
Application Title = ""
DLL Exports = "Include File Symbols"
DLL Import Library Choice = "Gen Lib For Current Mode"
Use VXIPNP Subdirectories for Import Libraries = False
Use Dflt Import Lib Base Name = True
Where to Copy DLL = "Do not copy"
Add Type Lib To DLL = False
Include Type Lib Help Links = False
Type Lib FP File = ""
Type Lib Guid = ""
Instrument Driver Support Only = False

[External Compiler Support]
Create UIR Callbacks File = False
Using LoadExternalModule = False

Create Project Symbols File = True
UIR Callbacks Obj File = ""
Project Symbols H File = ""
Project Symbols Obj File = ""

[DLL Debugging Support]
External Process Path = ""

D.1.4 Listing of host.c

```

/*****
*
* SVID.C - This is the main routine for the IRL Spectral Interface display
*          program. That is to say... START HERE!!!
*
* main().....The main camera program
*
*
*          Rewritten at the University of Calgary by Anna Johnson to compile under
*          a National Instruments CVI ANSI C compiler under Windows 9x, and to
*          control a 128 by 128 array at the Rothney Astrophysical Observatory.
*          Date of rewrite 23 March 2001
*          NOTE: Some modules (specifically the 16-bit PCI routines) were
*          rewritten using in-line assembler and were compiled using Microsoft
*          Visual C++ Version 5.0. Various Microsoft libraries were also referenced
*          so if you work with this code you will need both compilers. The GUIs
*          are generated using the CVI compiler.
*          NOTE: A 32-bit VXD was obtained from Spectral Instruments. This driver
*          requires a DLL and the VXD must be in C:\WINDOWS\SYSTEM as it is
dynamically
*          loaded. If you mess with this driver be aware that CVI requires the old-style
*          API calls, and cannot use the new non-compatible Microsoft calls. However,
there
*          does exist a facility in CVI for blowing off a new *.LIB file if you have the
*          DLL and the include file. See the CVI Programmer's Guide for further details.
*
*
*
* $Id: svid.c 1.0 1996/03/27 KRS Exp $
* $Id: svid.c 2.0 2001/03/23 SEJ Exp $
* $Id: svid.c 2.01 2001/05/17 SEJ Exp $
*****/
/

/*****
* INCLUDES & DEFINES
*****/
/

```

```

#define SVID_C

#include <utility.h>
#include <rs232.h>
#include <ansi_c.h>
#include <formatio.h>
#include <userint.h>
#include <ctype.h>
#include <stdio.h>
#include <stdarg.h>
#include <stddef.h>
#include <math.h>
#include <string.h>
#include <time.h>

#ifndef Bool
#define Bool unsigned char
#define TRUE 1
#define FALSE 0
#endif

/* Truth, lies, and other delusions */

/* stuff to make the file handlers happy */
#define CARD_SIZE 80 /* shades of the IBM 1620... */

/* stuff to define the Spectral Systems card */
#define DEVICE_ID 0x80B6
#define VENDOR_ID 0x10E8
#define INDEX_ID 0x0000

#define TIMEOUT_TIME (double)10
#define LIGHT_TIME (double)2

/* stuff associated with the 8255 parallel port for the filter wheel */
#define PORT_A_OFFSET 0
#define PORT_B_OFFSET 1
#define PORT_C_OFFSET 2
#define PORT_D_OFFSET 3
#define PORT_CONTROL_MASK 0x8B

#include "global.h" /* global defines and variables */
#include "svid.h"
#include "xlate.h" /* translates some ANSI calls, not in the CVI

```

```

library, to their CVI equivalents */
#include "siw95dll.h"      /* Spectral Systems API interface to DLL */
#include "fitslib.h"       /* FITS library include file as modified with wrappers */
#include "access32.h"      /* ACCESS I/O parallel port include file */
#include "startup.h"       /* CVI generated include file for GUI for startup screen */
#include "main.h"          /* CVI generated include file for GUI for main
menu */
#include "filemenu.h"      /* CVI generated include file for GUI for file
control menu */
#include "debugmenu.h"     /* CVI generated include file for GUI for debug
menu */
#include "errmenu.h"       /* CVI generated include file for GUI for error message
window */
#include "imagemenu.h"     /* CVI generated include file for GUI for image
manipulation window */
#include "setupmenu.h"     /* CVI generated include file for GUI for setup
menu */
#include "filtermenu.h"    /* CVI generated include file for GUI for filter
wheel control menu */
#include "about.h"         /* CVI generated include file for GUI for about
screen */

```

```

/* don't mess with the order of these defines. They must correspond with the order of
creation of the windows */

```

```

#define ERROR -1
#define MAIN_SOURCE 1 /* used as selector in
background loop portion of message cracker */
#define STARTUP_SOURCE 2 /* never called, but it has to
be here */
#define DEBUG_SOURCE 3 /* debug and low-level functions */
#define FILE_SOURCE 4 /* file open and close */
#define ERROR_SOURCE 5 /* handler for error window */
#define IMAGE_SOURCE 6 /* handler for image displays */
#define SETUP_SOURCE 7 /* handler for setup window */
#define FILTER_SOURCE 8 /* handler for filter window */
#define ABOUT_SOURCE 9 /* handle for about window */

```

```

/*****

```

```

* program strategy - just so you know.
* The main routine does all the works, but it is a background loop in a
* classic real-time system. What it does depends on flags that it runs
* in to which have been set by callbacks invoked by the GUI. This is to

```

```

* say, if the operator wants to do something, he clicks on an item. This
* results in a flag being set (and a value being changed, mayhaps). The
* flag is interrogated in the fullness of the time by the background loop
* which dispatches a task to do the operation requested.
*
* There are two sets of nested switches, although if you look at main()
* you will see only one. The first switch, which is in main(), pulls an
* event out of the event queue and determines only which menu generated
* the event. It then invokes a message cracker specific for that menu.
* The message cracker for the menu pulls the event flag out of the queue
* and dispatches tasks according to what the event flag is. The cracker
* will do other stuff as well as required to support the dispatched tasks.
* All inter-task communication is done through an object known as Block.
* There is a pointer to Block (named block, what else?) that everything gets
* passed to it. All the goodies are in the typedef struct DATABLOCK.
*
* Regarding communication with the 8255 PIO in the digital I/O card. The
* port assignments are as follows: Port A, 8 bits out, bit 0 LSB. Bits 2,1,0
* are the filter number to be turned to. Bit 3 is the command bit. Port B, 8 bits
* in. Bits 2,1,0 are the filter number where the wheel is now. Bit 4 is used
* to indicate the wheel is moving. Port C, 8 bits in. Bits 7,6,5,4 MSB, bits 3,2,1,0
* LSB, packed BCD for the detector temperature.
*
* Because the temperature has to update once per second, the driver for this display
* is put in with the clock update. It doesn't change often so shouldn't have to
* update often. If it overranges then it just sticks at 100 K. But you don't want
* it writing continuously to the display because this takes a long time to do and
* is a resource drain. Better that it be in the clock update and run only once
* per second.
* Also, the filter wheel stuff is updated on the screen only when there is a change for
* the same reason.
*
*
*****/
void main (void)
{
    int          m_icode;    /* return code from setup routines */
    int          source;     /* comes through circular ring, marks which menu
generated the command */
    int          incntl;     /* marks which control was toggled - used in crackers
referenced by switch */
    int          work;       /* standard work variables */

```

```

int          work2;

DATABLOCK    Block;          /* datablock, gets passed around between
various routines */
DATABLOCK    *block;         /* the pointer that allows this to
happen */

/* on entry, must set up pointers, as everything else works out of these objects */
block = &Block;              /* set pointer
to the data block */

/* initial system setup */
/* this is all there is to the initial system setup.
* The user has the opportunity to set the working
* directory, and to read (or store) a new configuration
*/
m_ecode = init_windows(block);      /* open all the screens
*/
m_ecode = init_spectral(block);      /* set up the Spectral
Instruments card */
m_ecode = allocate_buffers(block);  /* set up the video buffers */
m_ecode = get_home_dir(block);      /* find out where the
home directory is */
m_ecode = init_object(block);       /* initialize the block
object */
m_ecode = config_read(block);       /* read the
configuration file */
/* end initial system setup */

/*****
*
* BEGIN BACKGROUND LOOP
*
*****/
m_ecode = 0;
while(m_ecode == 0)
{
    send_serial(block);            /* Send a
character out */

```

```

        receive_serial(block);                /* get a
character in */
        crack_incoming(block);                /* run the
serial port message cracker */
        run_dma(block);                       /* run
the DMA transfer routine */
        run_utc_clock(block);                 /* get the
system date and time and display the clock */
        run_filterwheel(block);               /* run the
filter wheel control routine */

    if(GetUserEvent(0,&source,&incntl)== TRUE)
    {
        switch(source)
        {
            case MAIN_SOURCE:
            {
                m_icode = main_cracker(incntl, block);
                break;
            }
            case DEBUG_SOURCE:
            {
                m_icode = debug_cracker(incntl, block);
                break;
            }
            case FILE_SOURCE:
            {
                m_icode = file_cracker(incntl, block);
                break;
            }
            case ERROR_SOURCE:
            {
                m_icode = error_cracker(incntl, block);
                break;
            }
            case IMAGE_SOURCE:
            {
                m_icode = image_cracker(incntl, block);
                break;
            }
            case SETUP_SOURCE:
            {
                m_icode = setup_cracker(incntl, block);

```

```

        break;
    }
    case FILTER_SOURCE:
    {
        m_icode = filter_cracker(incntl, block);
        break;
    }
    case ABOUT_SOURCE:
    {
        m_icode = about_cracker(incntl, block);
        break;
    }
    default:
    {
        error_message(block, "FATAL ERROR - BAD SELECT IN
MAIN SWITCH");
        break;
    }
}

}

/*****
 *
 * END BACKGROUND LOOP
 *
 *****/

if(m_icode == EXIT)
{
    return;
}

printf("\n ABEND OCCURRED, ERROR CODE IS %d",m_icode);
printf("\n WINDOW CODE IS %d, CONTROL CODE IS %d",source,incntl);
return;
}

/*****
 *
 * THIS IS THE END OF THE MAIN ROUTINE
 *
 *****/

```

```

/*****
 *
 *   Filter Wheel Handler
 *
 *****/

int    run_filterwheel(DATABLOCK *block)
{

    unsigned short invalue;
    unsigned short us_work;

    invalue = InPort(block->port_base + PORT_B_OFFSET);    /*
    Read the port */
    us_work = invalue;
                    /* save for power-on check */

    invalue = invalue & 0x0004;
                    /* mask, leaving only the run flag */
    if(invalue == FALSE)
    {
        SetCtrlVal(block->mainmenu, MAIN_filter_power, FALSE);    /* if
    hardware sez filter is not moving, put that up */
    }
    else
    {
        SetCtrlVal(block->mainmenu, MAIN_filter_power, TRUE);    /* if
    hardware sez filter is moving, put that up */
    }

    us_work = us_work & 0x0008;
                    /* mask, leaving the power-on flag */
    if(us_work == FALSE)
    {
        SetCtrlAttribute(block->mainmenu,
        MAIN_campwr,ATTR_ON_COLOR,VAL_RED);
    }
    else
    {
        SetCtrlAttribute(block->mainmenu,
        MAIN_campwr,ATTR_ON_COLOR,VAL_GREEN);
    }
}

```

```

    }

    if(block->filter_moving == FALSE)
    /* if we are not commanding a filter movement */
    {
        /* then force a stop and exit */
        DeassertFilterPower(block);
        return(0);
    }
    if(block->filter_moving == TRUE)
    /* if we are commanding a filter movement */
    {
        /* then assert the command line */
        AssertFilterPower(block);
    }

    invalue = InPort(block->port_base + PORT_B_OFFSET);          /* Get
the current filter position number */
    if( (invalue & 0x000f) == 1);
    /* if it is the fiducial, then light the home light */
    {
        block->home_found = TRUE;
        block->active_filter_number = 1;
        SetCtrlVal(block->filtermenu, FILTER_home_found_led, TRUE);
    }

    if( (invalue & 0x000f) != block->active_filter_number)        /* if the active
filter number has changed */
    {
        block->active_filter_number = (invalue & 0x000f);        /* get the new
active filter number */
        ClearFilterIndicators(block);
        /* clear the lights */
        SetActiveFilterIndicator(block);
        /* show the new filter number light */
    }

    if(block->active_filter_number == block->active_filter_target) /* If we have found
the target */
    {
        block->filter_moving == FALSE;
        /* turn off the logic */
    }

```

```

        SetCtrlAttribute(block->imagemenu, IMAGE_bump_filter,
        ATTR_CMD_BUTTON_COLOR, VAL_LT_GRAY);
        /* make sure that the button in the image menu is reset */
        DeassertFilterPower(block);
        /* and pass this on to the hardware immediately */
    }

    return(0);
}

/* service routines for filter wheel handler */
void DeassertFilterPower(DATABLOCK *block)
{
    unsigned short outvalue;

    outvalue = block->active_filter_target;
    outvalue = outvalue & 0x0007;
    OutPort(block->port_base + PORT_A_OFFSET, outvalue);
    return;
}

void AssertFilterPower(DATABLOCK *block)
{
    unsigned short outvalue;

    outvalue = block->active_filter_target;
    outvalue = outvalue & 0x0007;
    outvalue = outvalue | 0x0008;
    OutPort(block->port_base, outvalue);
    return;
}

void ClearFilterIndicators(DATABLOCK *block)
{
    /* first, clear the set in the filter wheel menu */
    SetCtrlVal(block->filtermenu, FILTER_blank_led, FALSE);
    SetCtrlVal(block->filtermenu, FILTER_empty_led, FALSE);
    SetCtrlVal(block->filtermenu, FILTER_filter_1_led, FALSE);
    SetCtrlVal(block->filtermenu, FILTER_filter_2_led, FALSE);
    SetCtrlVal(block->filtermenu, FILTER_filter_3_led, FALSE);
    SetCtrlVal(block->filtermenu, FILTER_filter_4_led, FALSE);

```

```

/* now, clear the set in the image menu */
SetCtrlAttribute(block->imagemenu,IMAGE_f1,ATTR_CMD_BUTTON_COLOR,VAL
_WHITE);
SetCtrlAttribute(block->imagemenu,IMAGE_f2,ATTR_CMD_BUTTON_COLOR,VAL
_WHITE);
SetCtrlAttribute(block->imagemenu,IMAGE_f3,ATTR_CMD_BUTTON_COLOR,VAL
_WHITE);
SetCtrlAttribute(block->imagemenu,IMAGE_f4,ATTR_CMD_BUTTON_COLOR,VAL
_WHITE);
SetCtrlAttribute(block->imagemenu,IMAGE_f5,ATTR_CMD_BUTTON_COLOR,VAL
_WHITE);
SetCtrlAttribute(block->imagemenu,IMAGE_f6,ATTR_CMD_BUTTON_COLOR,VAL
_WHITE);

return;
}

void SetActiveFilterIndicator(DATABLOCK *block)
{
switch(block->active_filter_number)
{
case 1:
{
SetCtrlVal(block->filtermenu,FILTER_blank_led, TRUE);
SetCtrlAttribute(block->imagemenu,IMAGE_f1,ATTR_CMD_BUTTON
_COLOR,VAL_RED);
break;
}
case 2:
{
SetCtrlVal(block->filtermenu,FILTER_empty_led, TRUE);
SetCtrlAttribute(block->imagemenu,IMAGE_f2,ATTR_CMD_BUTTON
_COLOR,VAL_RED);
break;
}
case 3:
{
SetCtrlVal(block->filtermenu,FILTER_filter_1_led,
TRUE);
SetCtrlAttribute(block->imagemenu,IMAGE_f3,ATTR_CMD_BUTTON
_COLOR,VAL_RED);
break;
}
}
}

```

```

        case 4:
        {
            SetCtrlVal(block->filtermenu,FILTER_filter_2_led,
TRUE);
            SetCtrlAttribute(block->imagemenu,IMAGE_f4,ATTR_CMD_BUTTON
_COLOR,VAL_RED);
            break;
        }
        case 5:
        {
            SetCtrlVal(block->filtermenu,FILTER_filter_3_led,
TRUE);
            SetCtrlAttribute(block->imagemenu,IMAGE_f5,ATTR_CMD_BUTTON
_COLOR,VAL_RED);
            break;
        }
        case 6:
        {
            SetCtrlVal(block->filtermenu,FILTER_filter_4_led,
TRUE);
            SetCtrlAttribute(block->imagemenu,IMAGE_f6,ATTR_CMD_BUTTON
_COLOR,VAL_RED);
            break;
        }
    }
    return;
}

```

```

/*****
*
*   END FILTER WHEEL HANDLER
*
*****/

```

```

/*****
*
*   UTC Clock Handler
*
*****/

```

```

int run_utc_clock(DATABLOCK *block)

```

```

{

char  outbuffer[80];
int year;
int day;
int month;

int hours;
int minutes;
int seconds;

int utc_offset;
unsigned short work;
int temperature;
        /* making this a time-and-temperature sign */
unsigned long portaddress;
        /* hardware port address */

GetSystemDate( &month, &day, &year);
/* get local time */
GetSystemTime( &hours, &minutes, &seconds);

        /* Display is intensive. So, we want to update only on
        * a seconds change. The following will do it
        */
if(seconds == block->oldseconds)
{
    return(0);
}
block->oldseconds = seconds;

/* temperature handler */
portaddress = block->port_base + PORT_C_OFFSET; /* get temperature port
address */
block->port_c = InPort(portaddress); /* read port */
work = block->port_c & 0x000f; /* move over LSB */
block->port_c = block->port_c/16; /* shift right 4 */
work += block->port_c * 10; /* mult by 10 and add
to convert from packed BCD to hex */
SetCtrlVal(block->mainmenu,MAIN_temp,work); /* display temperature */

```

```

GetCtrlVal(block->imagemenu,IMAGE_utc_offset,&utc_offset);      /* get offset */

hours = hours - utc_offset;
    /* the simple part - a negative offset will ADD to the local clock */
if(hours > 23 )
{
    hours = hours - 24;
    day ++;
}

if(hours <0)
{
    hours = hours + 24;
    day --;
}

switch (month)
    /* this is the calendar correction. It is really */
{
    /* ugly, but the Gregorial calendar is that way */
case 1:
    /* January has 31 days */
    {
        if(day == 32)
        {
            month++;
            day = 1;
        }
        if(day == 0)
        {
            month = 12;
            day = 31;
            year --;
        }
        break;
    }
case 2:
    /* February */
    {
        if((year%4)==0)

```

```

/* on non-leap years use this branch */
{
    if(day == 29)
    {
        month++;
        day = 1;
    }
    if(day == 0)
    {
        month = 1;
        day = 31;
    }
}
else
{
    /* on leap years use this branch */
    if(day == 30)
    {
        month++;
        day = 1;
    }
    if(day == 0)
    {
        month = 1;
        day = 31;
    }
}
break;
}
case 3:
{
    if(day == 32)
    {
        month++;
        day = 1;
    }
    if(day == 0)
    {
        month = 2;
        day = 28;
        if((year%4) == 0)    day++;
    }
}
break;

```

```
    }  
case 4:  
    {  
        if(day == 31)  
        {  
            month++;  
            day = 1;  
        }  
        if(day == 0)  
        {  
            month = 3;  
            day = 31;  
        }  
        break;  
    }  
case 5:  
    {  
        if(day == 32)  
        {  
            month++;  
            day = 1;  
        }  
        if(day == 0)  
        {  
            month = 4;  
            day = 30;  
        }  
        break;  
    }  
case 6:  
    {  
        if(day == 31)  
        {  
            month++;  
            day = 1;  
        }  
        if(day == 0)  
        {  
            month = 5;  
            day = 31;  
        }  
        break;  
    }
```

```
case 7:
{
    if(day == 32)
    {
        month++;
        day = 1;
    }
    if(day == 0)
    {
        month = 6;
        day = 30;
    }
    break;
}
case 8:
{
    if(day == 32)
    {
        month++;
        day = 1;
    }
    if(day == 0)
    {
        month = 7;
        day = 31;
    }
    break;
}
case 9:
{
    if(day == 31)
    {
        month++;
        day = 1;
    }
    if(day == 0)
    {
        month = 8;
        day = 31;
    }
    break;
}
case 10:
```

```

    {
    if(day == 32)
        {
            month ++;
            day = 1;
        }
    if(day == 0)
        {
            month = 9;
            day = 30;
        }
    break;
    }
case 11:
    {
    if(day == 31)
        {
            month ++;
            day = 1;
        }
    if(day == 0)
        {
            month = 10;
            day = 31;
        }
    break;
    }
case 12:
    {
    if(day == 32)
        {
            month = 1;
            day = 1;
            year ++;
        }
    if(day == 0)
        {
            month = 11;
            day = 30;
        }
    break;
    }
}

```

/* so now we

```

have the corrected calendar. */

sprintf(outbuffer,"YR %d MO %d DY %d",year,month,day);
/* this is the display */
SetCtrlVal(block->mainmenu,MAIN_UTC_date,outbuffer);
sprintf(outbuffer,"HOUR %2d MIN %2d SEC %2d",hours, minutes, seconds);
SetCtrlVal(block->mainmenu,MAIN_UTC_time,outbuffer);

sprintf(block->timehack,"%04d-%02d-%02dT%02d:%02d:%02d",year,month,day,hours,
minutes,seconds); /* make up the FITS timestamp */

return(0);
}

/*****
 *
 * Spectral Instruments serial port handlers
 *
 *****/

int send_serial(DATABLOCK *block)
{
char whiteratbuffer [80];

if(block->whiterat == TRUE)
/* first, process the white rat */
{
if((block->outarm)==TRUE)
{
sprintf(whiteratbuffer,"\n\rFROM HOST, HEX VAL %x",*block->outptr);
ComWrt(1,whiteratbuffer,strlen(whiteratbuffer));
}
}

if((block->outarm)==TRUE)
/* then, process the SS card */
{
send_char(*(block->outptr)); /* send char to Spectral Systems card */
}
}

```

```

        block->outptr++;
        block->outcount--;
        if(block->outcount==0)
        {
            block->outarm = FALSE;
            block->outptr=block->outbuffer;
        }
    }
    return(0);
}

int  receive_serial(DATABLOCK *block)
{
    char  whiteratbuffer[80];

    if(chars_avail()!=0)
    {
        *(block->inptr)=get_char();

        if(block->whiterat == TRUE)
        {
            sprintf(whiteratbuffer,"%x",*(block->inptr));
            ComWrt(1,whiteratbuffer,strlen(whiteratbuffer));
        }

        block->inptr++;
        block->incount++;
        block->inarm=TRUE;
    }
    return(0);
}

int crack_incoming (DATABLOCK *block)
{
    char  whiteratbuffer[80];
    char  tmpchr;
    int    work;

```



```

SetCtrlAttribute(block->mainmenu,MAIN_donelight,ATTR_LABEL_VISIBLE,TRUE);

SetCtrlAttribute(block->mainmenu,MAIN_light_timer,ATTR_INTERVAL,LIGHT_TIMER);

SetCtrlAttribute(block->mainmenu,MAIN_light_timer,ATTR_ENABLED,TRUE);
SetCtrlAttribute(block->mainmenu,MAIN_timeout_timer,
ATTR_ENABLED, FALSE);
return(0);
}
tmpchr = block->inbuffer[3];
/* put a 0x00 so can use strcmp */
block->inbuffer[3]=0x00;
/* but save char - may need later */
if(block->whiterat == TRUE)
{
block->inbuffer[0] &= 0x7F;
/* Hazeltine 1500 sends strange stuff */
block->inbuffer[1] &= 0x7F;
block->inbuffer[2] &= 0x7F;
}
if(strcmp(block->inbuffer,"ERR")==FALSE)
/* if ERR came back light button */
{
SetCtrlVal(block->mainmenu,MAIN_errlight,TRUE);

SetCtrlAttribute(block->mainmenu,MAIN_errlight,ATTR_LABEL_VISIBLE,TRUE);
SetCtrlAttribute(block->mainmenu,MAIN_timeout_timer,
ATTR_ENABLED, FALSE);
SetCtrlAttribute(block->mainmenu,MAIN_timeout_timer,
ATTR_ENABLED, FALSE);
}
if(strcmp(block->inbuffer,"DON")==FALSE)
/* if DON came back then normal return */
{
/* so light button but start
timer */
SetCtrlVal(block->mainmenu,MAIN_donelight,TRUE);

SetCtrlAttribute(block->mainmenu,MAIN_donelight,ATTR_LABEL_VISIBLE,TRUE);

SetCtrlAttribute(block->mainmenu,MAIN_light_timer,ATTR_INTERVAL,LIGHT_TIMER

```

```

E);

SetCtrlAttribute(block->mainmenu,MAIN_light_timer,ATTR_ENABLED,TRUE);
SetCtrlAttribute(block->mainmenu,MAIN_timeout_timer,
ATTR_ENABLED, FALSE);

SetCtrlAttribute(block->mainmenu,MAIN_abort,ATTR_DIMMED,TRUE);
    }
    if(strcmp(block->inbuffer,"TST")==FALSE)
        /* if TST came back is normal but */
        {
            /* require manual reset so no
timer */
SetCtrlVal(block->mainmenu,MAIN_tstlight,TRUE);

SetCtrlAttribute(block->mainmenu,MAIN_tstlight,ATTR_LABEL_VISIBLE,TRUE);
SetCtrlAttribute(block->mainmenu,MAIN_timeout_timer,
ATTR_ENABLED, FALSE);
    }
    block->inarm = FALSE;
    /* in all cases, reset incoming */
    block->incount = 0;
    /* ticktock */
    block->inptr = block->inbuffer;
    return(0);
}
}
return(0);
}

/*****
*
*   Spectral Instruments DMA handler
*
*****/

int run_dma(DATABLOCK *block)
{
WORD      *dmaptr;
int        cols;

```

```

int         rows;

if(block->dma_running == TRUE)
{
    switch(dma_done())
    {
        case DMA_TRANSFER_IN_PROGRESS:
        {
            return(0);
            break;
        }
        case DMA_ERROR:
        {

SetCtrlAttribute(block->mainmenu,MAIN_dma_led,ATTR_OFF_COLOR,VAL_RED);
            end_dma();
            block->dma_running = FALSE;
            return(ERROR);
            break;
        }
        case DMA_COMPLETE:
        {

SetCtrlAttribute(block->mainmenu,MAIN_dma_led,ATTR_OFF_COLOR,VAL_GREEN
);
            end_dma();
            block->dma_running = FALSE;

            if(block->frame1_active)
/* copy over time hacks */
            {
                strcpy(block->timehack,block->timestamp1);
            }

            if(block->frame2_active)
/* copy to frame buffer */
            {
                strcpy(block->timehack,block->timestamp2);
            }
            break;
        }
        default:
        {

```

```

        error_message(block,"DMA TRANSFER ERROR,");
    }
}
return(SUCCESS);
}

/*****
 *
 *   MENU CALLBACK EVENT HANDLERS ARE HERE
 *
 *****/

int CVICALLBACK light_timer (int panel, int control, int event, void *ptr1, int dum1, int
dum2)
{
    /* This is cheating. This
    makes use */

    /* of the fact that the timer is
    on the */

    /* same panel (MAIN) as the
    light. */

    /* Beware of this if you
    change the code */
    SetCtrlVal(panel,MAIN_donelight,FALSE);
    /* shut off the light */
    SetCtrlAttribute(panel,MAIN_donelight,ATTR_LABEL_VISIBLE,FALSE);
    SetCtrlAttribute(panel,MAIN_light_timer,ATTR_ENABLED,FALSE);
    /* turn the timer off */
    return(0);
}

int CVICALLBACK timeout_timer(int panel, int control, int event, void *ptr1, int dum1,
int dum2)

    /* same trick as above */

{

```

```

SetCtrlVal(panel,MAIN_timeout,TRUE);
SetCtrlAttribute(panel,MAIN_timeout,ATTR_LABEL_VISIBLE,TRUE);
SetCtrlAttribute(panel,MAIN_abort,ATTR_DIMMED,TRUE);

SetCtrlAttribute(panel, MAIN_timeout_timer, ATTR_ENABLED, FALSE);

return(0);
}

void CVICALLBACK cbhist(int menubar, int menuitem, void *block, int panel)
{
int status;
char workstr[80];

status = LaunchExecutable("explorer.exe hlp\\history.html");
if(status != FALSE)
{
    sprintf(workstr,"NO HIST FORKING. CODE IS %d", status);
    error_message(block,workstr);
}
return;
}

void CVICALLBACK cbhelp(int menubar, int menuitem, void *block, int panel)
{
int status;
char workstr[80];

status=LaunchExecutable("explorer.exe hlp\\index.html");
if(status != FALSE)
{
    sprintf(workstr,"NO HELP FORKING. CODE IS %d",status);
    error_message(block,workstr);
}
return;
}

void CVICALLBACK cbabout(int menubar, int menuitem, void *voidptr, int panel)
{
DATABLOCK      *block;
block = (DATABLOCK *)voidptr;
DisplayPanel(block->aboutmenu);
}

```

```

return;
}

void CVICALLBACK cblegal(int menubar, int menuitem, void *block, int panel)
{
    int status;
    char workstr[80];

    status = LaunchExecutable ("explorer.exe hlp\\legal.html");
    if(status != FALSE)
    {
        sprintf(workstr,"NO LEGAL FORKING. CODE IS %d",status);
        error_message(block, workstr);
    }
    return;
}

/*****
*
* Message crackers follow. These are the dispatch routines. They
* are called by the switch in main().
*
*****/

/*****
*
* Message cracker for MAIN menu
*
*****/

int main_cracker(int event, DATABLOCK *block)
{
    int source;
    int m_icode;
    int work;

    switch(event)
    {
        case MAIN_exit:
            {
                close_program(block);
                return(EXIT);
            }
            /* main: exit */
    }
}

```

```

        break;
    }
    case MAIN_file_menu: /* main: start file
menu */
    {
        DisplayPanel(block->filemenu);
        break;
    }
    case MAIN_debug_menu: /* main: start debug
menu */
    {
        DisplayPanel(block->debugmenu);
        break;
    }
    case MAIN_image_menu: /* main: start image
handling menu */
    {
        DisplayPanel(block->imagemenu);
        break;
    }
    case MAIN_setup_menu:
    {
        DisplayPanel(block->setupmenu);
        break;
    }
    case MAIN_filter_menu:
    {
        DisplayPanel(block->filtermenu);
        break;
    }
    case MAIN_rst_buffers:
    {
        clear_buff();
        block->inarm = FALSE;
        block->incount = 0;
        block->inptr = block->inbuffer;
        block->outarm = FALSE;
        block->outcount = 0;
        block->outptr = block->outbuffer;

        /* next,
need to toggle the DONE light so that the user

```

knows that something actually did happen.

*/

```

        SetCtrlVal(block->mainmenu,MAIN_donelight,TRUE);

SetCtrlAttribute(block->mainmenu,MAIN_donelight,ATTR_LABEL_VISIBLE,TRUE);

SetCtrlAttribute(block->mainmenu,MAIN_light_timer,ATTR_INTERVAL,LIGHT_TIMER);

SetCtrlAttribute(block->mainmenu,MAIN_light_timer,ATTR_ENABLED,TRUE);
        SetCtrlAttribute(block->mainmenu,MAIN_timeout_timer,
ATTR_ENABLED, FALSE);

        break;
    }
    case MAIN_campwr:                                /* MAIN: toggle
camera power */
    {
        GetCtrlVal(block->mainmenu,MAIN_campwr,&work);
        if(work == 1 )
        {
            m_ecode = send_con(block);                /* turn camera power ON */
        }
        else
        {
            m_ecode = send_cof(block);                /* turn camera power OFF */
        }
        break;
    }
    case MAIN_cntlpwr:                                /* MAIN: toggle
controller power */
    {
        GetCtrlVal(block->mainmenu,MAIN_cntlpwr,&work);
        if(work == 1 )
        {
            m_ecode = send_pon(block);
        }
        else
        {
            m_ecode = send_pof(block);
        }
        break;
    }

```

```

    }
    case MAIN_shutter:                                /* main: start exposure */
    {
        SetCtrlAttribute(block->mainmenu,MAIN_abort,ATTR_DIMMED,FALSE);

        //          SetCtrlAttribute(block->mainmenu,MAIN_dma_led,
        ATTR_OFF_COLOR,VAL_YELLOW);
        SetCtrlVal          (block->mainmenu,MAIN_shutter,FALSE);
        block->dma_running = TRUE;
        do_dma(block);
        send_rrr(block);
        break;
    }
    case MAIN_abort:                                    /* main: abort exposure */
    {
        m_ecode=send_abr(block);

        SetCtrlAttribute(block->mainmenu,MAIN_abort,ATTR_DIMMED,TRUE);
        SetCtrlAttribute(block->mainmenu,MAIN_timeout_timer,
        ATTR_ENABLED, FALSE);
        break;
    }
    case MAIN_reset:                                    /* reset comm indicator lights
*/
    {
        SetCtrlAttribute(block->mainmenu,MAIN_errlight,
        ATTR_LABEL_VISIBLE, FALSE);
        SetCtrlAttribute(block->mainmenu,MAIN_errlight, ATTR_DIMMED,
        FALSE);
        SetCtrlAttribute(block->mainmenu,MAIN_tstlight,
        ATTR_LABEL_VISIBLE, FALSE);
        SetCtrlAttribute(block->mainmenu,MAIN_tstlight, ATTR_DIMMED,
        FALSE);
        SetCtrlAttribute(block->mainmenu,MAIN_timeout,
        ATTR_LABEL_VISIBLE, FALSE);
        SetCtrlAttribute(block->mainmenu,MAIN_timeout, ATTR_DIMMED,
        FALSE);
        SetCtrlVal(block->mainmenu,MAIN_errlight,FALSE);
        SetCtrlVal(block->mainmenu,MAIN_tstlight,FALSE);
        SetCtrlVal(block->mainmenu,MAIN_timeout,FALSE);

        SetCtrlAttribute(block->mainmenu,MAIN_timeout,ATTR_LABEL_VISIBLE, FALSE);

```

```

SetCtrlAttribute(block->mainmenu,MAIN_ouerrun,ATTR_LABEL_VISIBLE, FALSE);
        SetCtrlAttribute(block->mainmenu,MAIN_ouerrun,ATTR_DIMMED,
FALSE);
        SetCtrlVal(block->mainmenu,MAIN_ouerrun,FALSE);

SetCtrlAttribute(block->mainmenu,MAIN_abort,ATTR_DIMMED,TRUE);

        SetCtrlAttribute(block->mainmenu,MAIN_timeout_timer,
ATTR_ENABLED, FALSE);

SetCtrlAttribute(block->mainmenu,MAIN_dma_led,ATTR_OFF_COLOR,VAL_BLUE);

        block->incount=0;
        block->outcount=0;
        break;
    }
    default:                                     /* default trap */
    {
        error_message(block,"FEATURE NOT IMPLEMENTED THIS
VERSION");
        break;
    } /* takes care of stubs not yet implemented */
}
if(m_ecode == ERROR)
{
    SetCtrlVal(block->mainmenu,MAIN_ouerrun,TRUE);

SetCtrlAttribute(block->mainmenu,MAIN_ouerrun,ATTR_LABEL_VISIBLE,TRUE);
}
return(0);
}

/*****
*
* END MESSAGE CRACKER FOR MAIN MENU
*
*****/

```

```

/*****
*
* Message cracker for FILE menu
*
*****/

int file_cracker(int event, DATABLOCK *block)
{
int source;
int m_icode;
int work;

switch(event)
{
case FILEMENU_return:                                /* FILE: return */
{
HidePanel(block->filemenu);
return(0);
break;
}
case FILEMENU_write_config:                            /* FILE: write config
file */
{
GetTextBoxLineLength(block->filemenu, FILEMENU_config_file, 0,
&m_icode);
if(m_icode <= 0)
{
error_message(block, "WRITE_CONFIG: INVALID FILE
NAME");
return(0);
}
m_icode = GetTextBoxLine(block->filemenu, FILEMENU_config_file,
0, block->instr);
m_icode = config_write(block);
break;
}
case FILEMENU_read_config:                            /* FILE: read config
file */
{
GetTextBoxLineLength(block->filemenu, FILEMENU_config_file, 0,
&m_icode);
if(m_icode <= 0)

```



```

        {
            block->wddrive[0] -= 0x60;
        }
    else
        {
            block->wddrive[0] -= 0x40;
        }
    SetDrive((int)(block->wddrive[0]));
}

m_ecode = SetDir(block->wddir);
/* try to set the directory. If it fails */
if(m_ecode == FAILCODE)
/* then make the directory and set it
*/
    {
        MakeDir(block->wddir);
        SetDir(block->wddir);
    }
m_ecode = estab_dir(block);
/* after all of that, make the data dirs */
if(m_ecode == PASSED)
    {
        block->fileset_open = TRUE;
    }
    else
        {
            error_message(block,"ERROR ON CREATING
SUBDIRECTORIES");
        }
        break;
    }
default:
/* default trap */
    {
        break;
    }
/* should
never happen but we'll ignore that for the moment */
}
if(m_ecode == ERROR)
    {
        SetCtrlVal(block->mainmenu,MAIN_ouerrun,TRUE);
        SetCtrlAttribute(block->mainmenu,MAIN_ouerrun,ATTR_LABEL_VISIBLE,TRUE);
    }

```

```

    }
return(0);
}

```

```

/*****
 *
 * END MESSAGE CRACKER FOR FILE MENU
 *
 *****/

```

```

/*****
 *
 * Message cracker for ERROR menu
 *
 *****/

```

```

int error_cracker(int event, DATABLOCK *block)
{
int source;
int m_icode;
int work;

switch(event)
{
case ERRMSG_clear:                                /* error: return */
{
HidePanel(block->errormenu);
return(0);
break;
}
case ERRMSG_exit:                                /* error: exit right now */
{
HidePanel(block->errormenu);
close_program(block);
return(-1);
}
}
}

```

```

        }
    default:
        {}
    never happen but we'll ignore that for the moment */
}
return(0);
}

/* This is a service routine, not a cracker. However, it is a special case, so it is put here,
not elsewhere in
* the code. This routine displays the error message, as given to it by the routine which
had the error.
*/
void error_message(DATABLOCK *block, char *inmsg)
{
    DisplayPanel(block->errormenu);
    ResetTextBox(block->errormenu,ERRMSG_msg,inmsg);
    return;
}

/*****
*
* END MESSAGE CRACKER FOR ERROR MENU
*
*****/

/*****
*
* Message cracker for IMAGE menu
*
*****/

int image_cracker(int event, DATABLOCK *block)
{
    int    source;
    int    m_icode;

```

```

int    work;
int        work2;
int        work3;
WORD      *wordptr;
int        bytecolour;
int        colour;
FILE      *fps_indian;
char      b_inbyte;
long      dummy;
char      cs_workstr [80];
float     fwork;

switch(event)
{
    case IMAGE_bump_filter:                /* the filter lights are just
lights */
    {
        block->filter_moving = TRUE;      /* start the filter moving */
        block->active_filter_target++;     /* select the next filter */
        if(block->active_filter_target>6) block->active_filter_target = 1;

        SetCtrlAttribute(block->imagemenu,IMAGE_bump_filter,ATTR_CMD_BUTTON_COLOR,VAL_GREEN);
        break;
    }
    case IMAGE_return:                    /* IMAGE: return */
    {
        HidePanel(block->imagemenu);
        return(0);
        break;
    }
    case IMAGE_save_frame:                /* IMAGE: save the active
frame */
    {
        if(block->fileset_open == FALSE)
        {
            error_message(block,"ERROR - FILE SET HAS NOT BEEN
INITIALIZED");
            return(0);
        }
        m_ecode=MakeFits(block);
        break;
    }
}

```

```

case IMAGE_load_ramp:
{
if(block->frame1_active == TRUE) wordptr=block->dma1_array;
if(block->frame2_active == TRUE) wordptr=block->dma2_array;
work3=0;
for(work=0;work<NCOLS*NROWS;work++)
{
*wordptr=(WORD)work3;
wordptr++;
work3+=4;
}
break;
}
case IMAGE_send_sex: /* IMAGE: set the exposure
timer */
{
GetCtrlVal(block->imagemenu,IMAGE_exp_tim,&(block->exp_tim));
SetCtrlVal(block->debugmenu,DEBUG_exp_tim,block->exp_tim);
m_encode = send_sex(block);
break;
}
case IMAGE_auto_sel:
{
SetCtrlVal(block->imagemenu,IMAGE_auto_sel,FALSE);
error_message(block,"CANNOT AUTOLOAD POSITION: FUNCTION
NOT IMPLEMENTED");
return(0);
break;
}
case IMAGE_send_sb2: /* set video board bias */
/* this routine
assumes 10V bias voltage */
GetCtrlVal(block->imagemenu,IMAGE_bias, &work); /* get
the value in millivolts */
fwork = work;
/* convert to floating point */
fwork = fwork / 1000.0;
/* convert to volts */
fwork += 5.0;
/* add offset */
fwork = fwork / 10.0;
/* correct for range */
fwork = fwork * 4096;

```

```

/* convert to bits */
    work = fwork;
    /* convert to fixed point */
    work = work & 0x00000FFF;
/* ensure it is properly masked */
    block->bias = work;
/* save */
    send_sb2(block);
/* and send */
    break;
}
case IMAGE_send_sgn:          /* Set ADC gain. Since this word also picks
up the integration                                * speed and the Read
                                                    * display as well
                                                    */
{
    GetCtrlVal(block->imagemenu, IMAGE_gain, &work);
    switch(work)
/* this gets the gain */
    {
        case 1:      /* unity gain */
        {
            block->gain = 0x00000007;
            break;
        }
        case 2:      /* gain of 2.0 */
        {
            block->gain = 0x0000000B;
            break;
        }
        case 3:      /* gain of 4.75 */
        {
            block->gain = 0x0000000D;
            break;
        }
        case 4:      /* gain of 9.50 */
        {
            block->gain = 0x0000000E;
            break;
        }
    }
}

```

```

        GetCtrlVal(block->imagemenu,IMAGE_integrate,&work);      /* pick
up the integrate flag */
        switch(work)
        {
            case 0:      /* slow integration */
            {
                block->gain += 0x00000000;
/* this may seem strange, but it allows flexibility */
                break;
            }
            case 1:      /* fast integration */
            {
                block->gain += 0x00000100;
                break;
            }
        }

        GetCtrlVal(block->imagemenu,IMAGE_preamp,&work);          /* pick
up preamp output enable flag */
        switch(work)
        {
            case 0:      /* preamplifier output off */
            {
                block->gain += 0x00000000;
/* this may seem strange, but it allows flexibility */
                break;
            }
            case 1:      /* preamplifier output on */
            {
                block->gain += 0x00000400;
                break;
            }
        }

        send_sgn(block);
        break;
    }
case IMAGE_load_indian:
    /* load test pattern */

    /* doesn't have to be an indian */

```

```

bytes */
/* just has to have exactly 12288
{
  GetCtrlVal(block->setupmenu,SETUP_tpfile,cs_workstr);
  m_icode = GetFileInfo(cs_workstr,&dummy);
  /* first, check that file is available */
  if(m_icode != TRUE)
  {
    error_message(block,"CANNOT FIND TEST PATTERN");
    return(SUCCESS);
    /* returning a FAILCODE would be the right
    * thing to do, but it would kill the
program.
    * instead, return this and they can
read
    * the error message
    */
  }
  fps_indian = fopen(cs_workstr,"r");

  GetCtrlVal(block->imagemenu,IMAGE_framesel,&work);
  switch(work)
  {
    case 1:
    {
      wordptr=block->dma1_array;
      /* wipe the array */
      for(work=0;work<NROWS;work++)
      {
        for(work2=0;work2<NCOLS;work2++)
        {
          *wordptr = 0x00;
        }
      }

      wordptr=block->dma1_array;
      /* move down into image */
      wordptr+=2048;
      for(work=0;work<INDIAN_FILE_SIZE;work++)

```

```

        {
            b_inbyte = getc(fps_indian);
            bytecolour = (int)b_inbyte;
            bytecolour = bytecolour * 256;
            *wordptr = (unsigned short int)bytecolour;
            wordptr++;
        }
        break;
    }
    case 2:
    {
        wordptr=block->dma2_array;
        /* wipe the array */
        for(work=0;work<NROWS;work++)
        {
            for(work2=0;work2<NCOLS;work2++)
            {
                *wordptr = 0x00;
            }
        }

        wordptr=block->dma2_array;
        /* move down into image */
        wordptr+=2048;
        for(work=0;work<INDIAN_FILE_SIZE;work++)
        {
            b_inbyte = getc(fps_indian);
            bytecolour = (int)b_inbyte;
            bytecolour = bytecolour * 256;
            *wordptr = (unsigned short int)bytecolour;
            wordptr++;
        }
        break;
    }
}
fclose(fps_indian);
break;
}
case IMAGE_framesel:
/* IMAGE: select active
frame */
{
    GetCtrlVal(block->imagemenu,IMAGE_framesel,&work);
    switch(work)

```

```

        {
        case 1:
            {
                block->frame1_active = TRUE;
                block->frame2_active = FALSE;

SetCtrlVal(block->imagemenu,IMAGE_frame1_led,TRUE);

SetCtrlVal(block->imagemenu,IMAGE_frame2_led,FALSE);
                break;
            }
        case 2:
            {
                block->frame1_active = FALSE;
                block->frame2_active = TRUE;

SetCtrlVal(block->imagemenu,IMAGE_frame1_led,FALSE);

SetCtrlVal(block->imagemenu,IMAGE_frame2_led,TRUE);
                break;
            }
        }
        break;
    }
case IMAGE_display_frame: /* IMAGE: load and display active
frame */
    {
        GetCtrlVal(block->imagemenu,IMAGE_framesel,&work);
        switch(work)
        {
        case 1:
            {
                wordptr=block->dma1_array;
                for(work2=0;work2<NCOLS;work2++)
                {
                    for(work3=0;work3<NROWS;work3++)
                    {

PointSet(&(block->frame1point),work3,work2);
                                bytecolour = (int)(*wordptr)/256;
                                colour= (int)(bytecolour*256*256 +
bytecolour*256 + (bytecolour));
                                wordptr++;
                    }
                }
            }
        }
    }

```

```

SetCtrlAttribute(block->imagemenu,IMAGE_frame1,ATTR_PEN_COLOR,colour);
SetCtrlAttribute(block->imagemenu,IMAGE_frame1,ATTR_PEN_FILL_COLOR,colour);
CanvasDrawPoint(block->imagemenu,
IMAGE_frame1, block->frame1point);
    }
    }
SetCtrlAttribute(block->mainmenu,MAIN_dma_led,ATTR
_OFF_COLOR,VAL_BLUE);
break;
}
case 2:
{
wordptr=block->dma2_array;
for(work2=0;work2<NCOLS;work2++)
{
for(work3=0;work3<NROWS;work3++)
{
PointSet(&(block->frame2point),work3,work2);
bytecolour = (int)(*wordptr)/256;
colour= (int)(bytecolour*256*256 +
bytecolour*256 + (bytecolour));
wordptr++;
SetCtrlAttribute(block->imagemenu,IMAGE_frame2,ATTR_PEN_COLOR,colour);
SetCtrlAttribute(block->imagemenu,IMAGE_frame2,ATTR_PEN_FILL_COLOR,colour);
CanvasDrawPoint(block->imagemenu,
IMAGE_frame2, block->frame2point);
}
}
SetCtrlAttribute(block->mainmenu,MAIN_dma_led,ATTR
_OFF_COLOR,VAL_BLUE);
break;
}
break;
}
default:
{}
/* default trap */
/* should
never happen but we'll ignore that for the moment */

```

```

    }
    if(m_ecode == ERROR)
    {
        SetCtrlVal(block->mainmenu,MAIN_ouerrun,TRUE);

        SetCtrlAttribute(block->mainmenu,MAIN_ouerrun,ATTR_LABEL_VISIBLE,TRUE);
    }
    return(0);
}

```

```

/*****
 *
 * END MESSAGE CRACKER FOR IMAGE MENU
 *
 *****/

```

```

/*****
 *
 * Message cracker for SETUP menu
 *
 *****/

```

```

int setup_cracker(int event, DATABLOCK *block)
{
    int source;
    int m_ecode;
    int work;

```

this can do is return. It loads strings and stuff for the FITS
only control is the return.

```

/* note: all
 * files, but the
 */

```

```

switch(event)
{
    case SETUP_return:
        {
            HidePanel(block->setupmenu);
            return(0);
        }
}
/* setup: return */

```

```

        break;
    }
    case SETUP_init: /* setup: init filter port
number */
    {
        OutPort(block->port_base + PORT_D_OFFSET,
PORT_CONTROL_MASK);
        break;
    }
    default: /* default trap */
    {} /* should
never happen but we'll ignore that for the moment */
}
if(m_ecode == ERROR)
{
    SetCtrlVal(block->mainmenu,MAIN_overnun,TRUE);

    SetCtrlAttribute(block->mainmenu,MAIN_overnun,ATTR_LABEL_VISIBLE,TRUE);
}
return(0);
}

/*****
*
* END MESSAGE CRACKER FOR SETUP MENU
*
*****/

/*****
*
* MESSAGE CRACKER FOR DEBUG MENU
*
*****/

int debug_cracker(int event, DATABLOCK *block)
{
    int source;
    int m_ecode;
    int work;

```

```

unsigned int  address;

switch(event)
{
    case DEBUG_return:                /* DEBUG: return to main
menu */
    {
        HidePanel(block->debugmenu);
        break;
    }
    case DEBUG_send_tst:                /* DEBUG: send the test
message */
    {
        m_ecode = send_tst(block);
        break;
    }
    case DEBUG_send_err:                /* DEBUG: send the error
message */
    {
        m_ecode = send_err(block);
        break;
    }
    case DEBUG_send_sex:                /* DEBUG: set the exposure
timer */
    {
        GetCtrlVal(block->debugmenu,DEBUG_exp_tim,&(block->exp_tim));
        SetCtrlVal(block->imagemenu,IMAGE_exp_tim,block->exp_tim);
        m_ecode = send_sex(block);
        break;
    }
    case DEBUG_send_ldw:                /* DEBUG: send the latch
word */
    {
        GetCtrlVal(block->debugmenu,DEBUG_port_flags,&(block->port_flags));
        m_ecode = send_ldw(block);
        break;
    }
    case DEBUG_send_rst:                /* DEBUG: reboot the
controller */
    {
        m_ecode = send_rst(block);
        break;
    }
}

```

```

    }
    case DEBUG_send_osh:                /* DEBUG: open shutter */
    {
        m_icode = send_osh(block);
        break;
    }
    case DEBUG_send_csh:                /* DEBUG: close shutter */
    {
        m_icode = send_csh(block);
        break;
    }
    case DEBUG_send_wrm:                /* DEBUG: send write
memory */
    {
        m_icode = send_wrm(block);
        break;
    }
    case DEBUG_send_rdm:                /* DEBUG: send read
memory */
    {
        m_icode = send_rdm(block);
        break;
    }
    case DEBUG_institution:             /* DEBUG: get institution
code */
    {
        m_icode = send_institution(block);
        break;
    }
    case DEBUG_firmware:                /* get firmware version */
    {
        m_icode = send_firmware(block);
        break;
    }
    case DEBUG_sread:                  /* sequential read memory */
    {
        GetCtrlVal(block->debugmenu,DEBUG_address,&address);
        address++;
        if(address >= 0x10000) address = 0;
        SetCtrlVal(block->debugmenu,DEBUG_address,address);
        m_icode = send_rdm(block);
        break;
    }

```

```

case DEBUG_whiterat:                                /* get white rat status */
{
    work = block->whiterat;
    GetCtrlVal(block->debugmenu,DEBUG_whiterat,&(block->whiterat));
    work = block->whiterat;
    if((work == FALSE) && (block->whiterat == TRUE))
    {
        OpenComConfig(1,"COM1", 9600,0,8,1,0,0);
        sprintf(block->outstring,"\nWHITE RAT STARTED\n\r");
        ComWrt(1,block->outstring,strlen(block->outstring));
    }
    break;
}
default:                                             /* default trap */
{}                                                  /* this can
happen if the numerics are scrolled, so should not count as an error */
}
if(m_ecode == ERROR)
{
    SetCtrlVal(block->mainmenu,MAIN_overnun,TRUE);

    SetCtrlAttribute(block->mainmenu,MAIN_overnun,ATTR_LABEL_VISIBLE,TRUE);
}
if(m_ecode == ERROR)
{
    SetCtrlVal(block->mainmenu,MAIN_overnun,TRUE);

    SetCtrlAttribute(block->mainmenu,MAIN_overnun,ATTR_LABEL_VISIBLE,TRUE);
}
return(0);
}

/*****
*
*   END DEBUG MENU MESSAGE CRACKER
*
*****/

/*****
*
*   Message cracker for FILTER menu
*
*****/

```

```

*****

int filter_cracker(int event, DATABLOCK *block)
{
int source;
int m_ecode;
int work;

switch(event)
{
case FILTER_return:                                /* filter: return */
{
HidePanel(block->filtermenu);
return(0);
break;
}
case FILTER_stop_wheel:                            /* same thing as turn power
off */
{
block->filter_moving = FALSE;
SetCtrlVal(block->mainmenu, MAIN_filter_power, FALSE);
break;
}
case FILTER_seek_home:
{
block->active_filter_number = -99;
block->active_filter_target = 1;
SetCtrlVal(block->filtermenu, FILTER_home_found_led,
FALSE);
SetCtrlVal(block->filtermenu, FILTER_blank_led,
FALSE);
SetCtrlVal(block->filtermenu, FILTER_empty_led,
FALSE);
SetCtrlVal(block->filtermenu, FILTER_filter_1_led,
FALSE);
SetCtrlVal(block->filtermenu, FILTER_filter_2_led,
FALSE);
SetCtrlVal(block->filtermenu, FILTER_filter_3_led,
FALSE);
SetCtrlVal(block->filtermenu, FILTER_filter_4_led,
FALSE);
SetCtrlVal(block->filtermenu, FILTER_moving_led,
TRUE);
}
}
}

```

```

FALSE);
    SetCtrlVal(block->filtermenu, FILTER_stopped_led,
    block->filter_moving = TRUE;
    break;
}
case FILTER_seek_blank:
{
    if(block->active_filter_number == 1)
    {
        return(0);
    }
    block->active_filter_target = 1;
    block->filter_moving = TRUE;
    break;
}
case FILTER_seek_empty:
{
    if(block->active_filter_number == 2)
    {
        return(0);
    }
    block->active_filter_target = 2;
    block->filter_moving = TRUE;
    break;
}
case FILTER_seek_filter_1:
{
    if(block->active_filter_number == 3)
    {
        return(0);
    }
    block->active_filter_target = 3;
    block->filter_moving = TRUE;
    break;
}
case FILTER_seek_filter_2:
{
    if(block->active_filter_number == 4)
    {
        return(0);
    }
    block->active_filter_target = 4;
    block->filter_moving = TRUE;

```

```

        break;
    }
    case FILTER_seek_filter_3:
    {
        if(block->active_filter_number == 5)
        {
            return(0);
        }
        block->active_filter_target = 5;
        block->filter_moving = TRUE;
        break;
    }
    case FILTER_seek_filter_4:
    {
        if(block->active_filter_number == 6)
        {
            return(0);
        }
        block->active_filter_target = 6;
        block->filter_moving = TRUE;
        break;
    }
    default:
        {} /* default trap */
        {} /* should
never happen but we'll ignore that for the moment */
    }
    if(m_ocode == ERROR)
    {
        SetCtrlVal(block->mainmenu,MAIN_overrun,TRUE);

        SetCtrlAttribute(block->mainmenu,MAIN_overrun,ATTR_LABEL_VISIBLE,TRUE);
    }
    return(0);
}

/*****
*
* END MESSAGE CRACKER FOR FILTER MENU
*
*****/

```

```

/*****
 *
 * Message cracker for ABOUT menu
 *
 *****/

int about_cracker(int event, DATABLOCK *block)
{
int source;
int m_ecode;
int work;

/* note: all
this can do is return.*/

switch(event)
{
case ABOUT_return:          /* ABOUT: return */
{
HidePanel(block->aboutmenu);
return(0);
break;
}
default:                    /* default trap */
{}                          /* should
never happen but we'll ignore that for the moment */
}
return(0);
}

/*****
 *
 * END MESSAGE CRACKER FOR ABOUT MENU
 *
 *****/

```

```

/*****
*
*      BEGIN INITIALIZATION ROUTINES
*      These are the routines called in main() when doing the
*      initial system setup.
*
*****/

/*****
*
*  INIT_WINDOWS
*      This routine initializes the screens and opens the main
*      screen.
*
*****/

int init_windows (DATABLOCK *block)
{
    int          work, work1, work2;          /* standard work variables */
    Winhndl handle;                          /* a local window
    handle */
    Bool  fontsel;                          /* a toggle used in font
    selection */
    char  workstr          [81];             /* a work string */
    int   file_status;                      /* status return for file
    interrogation function */
    long  file_size;                       /* file size return for file
    interrogation function */
    int   file_handle;                      /* if we can access this file
    we have been here before
                                           * and can
    load screen initialization data from the file.
                                           * if we have
    not been here before, we need to run the
                                           * font
    selection routine.
                                           */
}

```

```

/* First, open and display the main menu, then open
 * and display the startup screen.
 * After all the rest of the screens load and the indicators are set up,
 * hide the startup screen. It will never be called again and represents a
 * memory leak, but at least the user will know the program didn't go off to
 * never-never land.
 */
block->mainmenu = LoadPanel(0,"main.uir" , MAIN);
DisplayPanel(block->mainmenu);
handle          = LoadPanel(block->mainmenu,"startup.uir", STARTUP);
DisplayPanel(handle);

/* then, open but do not display the rest of the menu system */
block->debugmenu = LoadPanel(block->mainmenu,"debugmenu.uir", DEBUG);
block->filemenu  = LoadPanel(block->mainmenu,"filemenu.uir", FILEMENU);
block->errormenu = LoadPanel(block->mainmenu,"errmenu.uir", ERRMSG);
block->imagemenu = LoadPanel(block->mainmenu,"imagemenu.uir", IMAGE);
block->setupmenu = LoadPanel(block->mainmenu,"setupmenu.uir", SETUP);
block->filtermenu = LoadPanel(block->mainmenu,"filtermenu.uir", FILTER);
block->aboutmenu = LoadPanel(block->mainmenu,"about.uir", ABOUT);

/* set up and clear canvases in imagemenu */
CanvasDefaultPen(block->imagemenu,IMAGE_frame1);
CanvasDefaultPen(block->imagemenu,IMAGE_frame2);
block->frame1_active = TRUE;
block->frame2_active = FALSE;

SetCtrlAttribute(block->imagemenu,IMAGE_frame1_led,ATTR_ON_COLOR,VAL_GREEN);
SetCtrlAttribute(block->imagemenu,IMAGE_frame1_led,ATTR_OFF_COLOR,VAL_RED);
SetCtrlAttribute(block->imagemenu,IMAGE_frame2_led,ATTR_ON_COLOR,VAL_GREEN);
SetCtrlAttribute(block->imagemenu,IMAGE_frame2_led,ATTR_OFF_COLOR,VAL_RED);
SetCtrlVal(block->imagemenu,IMAGE_frame1_led,TRUE);
SetCtrlVal(block->imagemenu,IMAGE_frame2_led,FALSE);

```

```

/* set up the interval timers */
SetCtrlAttribute(block->mainmenu,MAIN_timeout_timer,ATTR_INTERVAL,TIMEOUT_TIME);
SetCtrlAttribute(block->mainmenu,MAIN_timeout_timer,ATTR_ENABLED,FALSE);

SetCtrlAttribute(block->mainmenu,MAIN_light_timer,ATTR_INTERVAL,LIGHT_TIME);
SetCtrlAttribute(block->mainmenu,MAIN_light_timer,ATTR_ENABLED,FALSE);

SetCtrlAttribute(block->mainmenu,MAIN_timeout,ATTR_OFF_COLOR,VAL_LT_GRAY);
SetCtrlAttribute(block->mainmenu,MAIN_timeout,ATTR_ON_COLOR,VAL_RED);
SetCtrlAttribute(block->mainmenu,MAIN_timeout,ATTR_LABEL_BGCOLOR,VAL_RED);
SetCtrlAttribute(block->mainmenu,MAIN_timeout,ATTR_LABEL_COLOR,VAL_WHITE);
SetCtrlAttribute(block->mainmenu,MAIN_timeout,ATTR_LABEL_VISIBLE,FALSE);
SetCtrlVal          (block->mainmenu,MAIN_timeout,FALSE);

/* set up the indicator lights */
SetCtrlAttribute(block->mainmenu,MAIN_errlight,ATTR_OFF_COLOR,VAL_LT_GRAY);
SetCtrlAttribute(block->mainmenu,MAIN_errlight,ATTR_ON_COLOR,VAL_RED);
SetCtrlAttribute(block->mainmenu,MAIN_errlight,ATTR_LABEL_BGCOLOR,VAL_RED);
SetCtrlAttribute(block->mainmenu,MAIN_errlight,ATTR_LABEL_COLOR,VAL_WHITE);
SetCtrlAttribute(block->mainmenu,MAIN_errlight,ATTR_LABEL_VISIBLE,FALSE);
SetCtrlVal          (block->mainmenu,MAIN_errlight,FALSE);

SetCtrlAttribute(block->mainmenu,MAIN_tstlight,ATTR_OFF_COLOR,VAL_LT_GRAY);
SetCtrlAttribute(block->mainmenu,MAIN_tstlight,ATTR_ON_COLOR,VAL_YELLOW);
SetCtrlAttribute(block->mainmenu,MAIN_tstlight,ATTR_LABEL_BGCOLOR,VAL_YELLOW);
SetCtrlAttribute(block->mainmenu,MAIN_tstlight,ATTR_LABEL_COLOR,VAL_BLUE);

```

```

SetCtrlAttribute(block->mainmenu,MAIN_tstlight,ATTR_LABEL_VISIBLE,FALSE);
SetCtrlVal      (block->mainmenu,MAIN_tstlight,FALSE);

SetCtrlAttribute(block->mainmenu,MAIN_donelight,ATTR_OFF_COLOR,VAL_LT_G
RAY);
SetCtrlAttribute(block->mainmenu,MAIN_donelight,ATTR_ON_COLOR,
VAL_GREEN);
SetCtrlAttribute(block->mainmenu,MAIN_donelight,ATTR_LABEL_BGCOLOR,VAL_
BLACK);
SetCtrlAttribute(block->mainmenu,MAIN_donelight,ATTR_LABEL_COLOR,VAL_WH
ITE);
SetCtrlAttribute(block->mainmenu,MAIN_donelight,ATTR_LABEL_VISIBLE,FALSE);
SetCtrlVal      (block->mainmenu,MAIN_donelight,FALSE);

/* set the main menu indicator lights for initial status */
SetCtrlAttribute(block->mainmenu,MAIN_errlight,ATTR_LABEL_VISIBLE,FALSE);
SetCtrlAttribute(block->mainmenu,MAIN_errlight,ATTR_DIMMED,FALSE);
SetCtrlAttribute(block->mainmenu,MAIN_tstlight,ATTR_LABEL_VISIBLE,FALSE);
SetCtrlAttribute(block->mainmenu,MAIN_tstlight,ATTR_DIMMED,FALSE);
SetCtrlAttribute(block->mainmenu,MAIN_timeout,ATTR_LABEL_VISIBLE,FALSE);
SetCtrlAttribute(block->mainmenu,MAIN_timeout,ATTR_DIMMED,FALSE);
SetCtrlVal(block->mainmenu,MAIN_errlight,FALSE);
SetCtrlVal(block->mainmenu,MAIN_tstlight,FALSE);
SetCtrlVal(block->mainmenu,MAIN_timeout,FALSE);
SetCtrlAttribute(block->mainmenu,MAIN_timeout,ATTR_LABEL_VISIBLE,FALSE);

SetCtrlAttribute(block->mainmenu,MAIN_overnrun,ATTR_OFF_COLOR,VAL_LT_GR
AY);
SetCtrlAttribute(block->mainmenu,MAIN_overnrun,ATTR_ON_COLOR,VAL_RED);
SetCtrlAttribute(block->mainmenu,MAIN_overnrun,ATTR_LABEL_BGCOLOR,VAL_R
ED);
SetCtrlAttribute(block->mainmenu,MAIN_overnrun,ATTR_LABEL_COLOR,VAL_WHI
TE);
SetCtrlAttribute(block->mainmenu,MAIN_overnrun,ATTR_LABEL_VISIBLE,FALSE);
SetCtrlAttribute(block->mainmenu,MAIN_overnrun,ATTR_DIMMED,FALSE);
SetCtrlVal(block->mainmenu,MAIN_overnrun,FALSE);

/* power lights */
SetCtrlAttribute(block->mainmenu,MAIN_cntlpwr,ATTR_OFF_COLOR,VAL_RED);
SetCtrlAttribute(block->mainmenu,MAIN_cntlpwr,ATTR_ON_COLOR,VAL_GREEN);

SetCtrlAttribute(block->mainmenu,MAIN_campwr,ATTR_OFF_COLOR,VAL_YELLOW

```

```

W);
SetCtrlAttribute(block->mainmenu,MAIN_campwr,ATTR_ON_COLOR,VAL_YELLOW);
W);

/* dim the abort key in the shutter control */
SetCtrlAttribute(block->mainmenu,MAIN_abort,ATTR_DIMMED,TRUE);

/* set up indicator light for DMA status */
/* this will always be false. It will show several colours, depending on
 * what the DMA transfer is doing at the time.
 * Possible values are: BLUE:      dma not active
 *                          YELLOW:  dma active, not complete,
first transfer
 *                          YELLOW:  dma active, not complete,
second transfer
 *                          GREEN:    dma complete
 *                          RED:      dma error
 */
SetCtrlAttribute(block->mainmenu,MAIN_dma_led,ATTR_OFF_COLOR,VAL_BLUE);
SetCtrlVal      (block->mainmenu,MAIN_dma_led,FALSE);

/* set up the indicator lights for the filter menu */
SetCtrlAttribute(block->filtermenu, FILTER_home_found_led, ATTR_OFF_COLOR,
VAL_RED);
SetCtrlAttribute(block->filtermenu, FILTER_home_found_led, ATTR_ON_COLOR,
VAL_GREEN);

SetCtrlAttribute(block->filtermenu, FILTER_blank_led, ATTR_OFF_COLOR,
VAL_GRAY);
SetCtrlAttribute(block->filtermenu, FILTER_blank_led, ATTR_ON_COLOR,
VAL_GREEN);

SetCtrlAttribute(block->filtermenu, FILTER_empty_led, ATTR_OFF_COLOR,
VAL_GRAY);
SetCtrlAttribute(block->filtermenu, FILTER_empty_led, ATTR_ON_COLOR,
VAL_GREEN);

SetCtrlAttribute(block->filtermenu, FILTER_filter_1_led, ATTR_OFF_COLOR,
VAL_GRAY);
SetCtrlAttribute(block->filtermenu, FILTER_filter_1_led, ATTR_ON_COLOR,
VAL_GREEN);

```

```

SetCtrlAttribute(block->filtermenu, FILTER_filter_2_led, ATTR_OFF_COLOR,
VAL_GRAY);
SetCtrlAttribute(block->filtermenu, FILTER_filter_2_led, ATTR_ON_COLOR,
VAL_GREEN);

SetCtrlAttribute(block->filtermenu, FILTER_filter_3_led, ATTR_OFF_COLOR,
VAL_GRAY);
SetCtrlAttribute(block->filtermenu, FILTER_filter_3_led, ATTR_ON_COLOR,
VAL_GREEN);

SetCtrlAttribute(block->filtermenu, FILTER_filter_4_led, ATTR_OFF_COLOR,
VAL_GRAY);
SetCtrlAttribute(block->filtermenu, FILTER_filter_4_led, ATTR_ON_COLOR,
VAL_GREEN);

SetCtrlAttribute(block->filtermenu, FILTER_moving_led, ATTR_OFF_C
OLOR, VAL_GRAY);
SetCtrlAttribute(block->filtermenu, FILTER_moving_led, ATTR_ON_COLOR,
VAL_GREEN);

SetCtrlAttribute(block->filtermenu, FILTER_stopped_led, ATTR_OFF_C
OLOR, VAL_GRAY);
SetCtrlAttribute(block->filtermenu, FILTER_stopped_led, ATTR_ON_COLOR,
VAL_GREEN);

/* after all the above, load the menu bar for the main menu */
block->menuhandle = LoadMenuBar(block->mainmenu, "MAIN.UIR",MENU);
SetMenuBarAttribute(block->menuhandle,MENU_help,
ATTR_CALLBACK_DATA,block);
SetMenuBarAttribute(block->menuhandle,MENU_about,ATTR_CALLBACK_DATA,bl
ock);
SetMenuBarAttribute(block->menuhandle,MENU_legal,ATTR_CALLBACK_DATA,blo
ck);
SetMenuBarAttribute(block->menuhandle,MENU_hist,
ATTR_CALLBACK_DATA,block);

/* last thing to do is to get rid of the startup panel
*/
HidePanel(handle);
return(0);

}

```

```

/*****
 *
 * INIT_OBJECT
 *   Initializes the block object
 *
 *****/

int    init_object(DATABLOCK *block)
{

    /* the window handles do not need to be initialized, so ignore them. */

    strcpy(block->message,"STARTUP MESSAGE");
    strcpy(block->wname,"");
    strcpy(block->wddrive,"");
    strcpy(block->wddir,"");
    strcpy(block->wdfile,"");
    block->fileset_open = FALSE;

    block->homedrive = 3;                                /* set to drive
C */
    strcpy(block->homedir,"\\ircamera\\");                /* set to IRCAMERA */

    block->dma_running = FALSE;
    block->frame1_active = TRUE;
    block->frame2_active = FALSE;
    strcpy(block->timestamp1,"UNINITIALIZED TIMESTAMP");
    strcpy(block->timestamp2,"UNINITIALIZED TIMESTAMP");
    strcpy(block->timehack,"UNINITIALIZED TIMESTAMP");

    block->exp_tim = 0;
    block->port_flags = 0;
    block->bias = 0;
    block->gain = 0;

    block->outcount = 0;
    block->outptr = block->outbuffer;
    block->outarm = FALSE;

    block->return_expected = FALSE;                      /* no return from controller expected on
start */
    block->incount = 0;
    block->inptr = block->inbuffer;

```

```

block->inarm = FALSE;

block->flatctr = 0;
block->darkctr = 0;
block->backctr = 0;
block->imagectr = 0;

block->whiterat = FALSE;

block->home_found = FALSE;
block->active_filter_number = -99;
block->active_filter_target = -99;
block->filter_power = FALSE;
block->filter_moving = FALSE;

block->port_base = 0;

block->oldseconds = 0;

return(SUCCESS);
}

/*****
 *
 *  INIT_SPECTRAL
 *    Initializes the Spectral Instruments interface card
 *
 *****/

int  init_spectral(DATABLOCK *block)
{
    DWORD  retval;
    BYTE  err;

    block->inptr=block->inbuffer;
    block->outptr=block->outbuffer;
    block->dma_running = FALSE;

    retval = 99L;
    block->inptr=block->inbuffer;

```

```

block->outptr=block->outbuffer;
retval=install_isr(DEVICE_ID, VENDOR_ID, INDEX_ID);
err=(BYTE)retval & 0xFF;
if(err)
{
    sprintf(block->inbuffer,"SPECTRAL SYSTEMS BOARD ERROR. CODE IS
%d",err);
    error_message(block,block->inbuffer);
}

// if(block->whiterat == TRUE)
{
    OpenComConfig(1,"COM1", 9600,0,8,1,0,0);
    sprintf(block->outstring,"\nIN init_spectral. Install_isr return value is
%08x\n\r",retval);
    ComWrt(1,block->outstring,strlen(block->outstring));
    if(err)
    {
        sprintf(block->outstring,"\n\rERROR: SPECTRAL SYSTEMS BOARD
NOT FOUND. CODE IS %d\n\r",err);
        ComWrt(1,block->outstring,strlen(block->outstring));
        return(ERROR);
    }
}

/* put error trapping here */

block->outarm=FALSE;
block->inarm = FALSE;
block->incount = 0;
block->outcount=0;

init_com(BAUD, PARITY, BITS, STOP, BUFFERSIZE);
clear_buff();

return(retval);
}

```

```

/*****
* ALLOCATE_BUFFERS () - Allocate memory for the various program arrays.
*****/

/
int allocate_buffers (DATABLOCK *block)
{
int xx;

/* there are only two in this setup. Selection is controlled in the image submenu */

/* Allocate a buffer for the DMA frame1 */
if (!(block->dma1_array = malloc(NROWS * NCOLS * BYTES_PER_PIX)))
{
return(DMA_ARRAY1_ALLOCATION_FAILURE);
}

/* Allocate a buffer for the DMA frame2 */
if (!(block->dma2_array = malloc(NROWS * NCOLS * BYTES_PER_PIX)))
{
return(DMA_ARRAY2_ALLOCATION_FAILURE);
}

for (xx=0; xx<NROWS*NCOLS; xx++)
{
*(block->dma1_array) = 0x0000;
*(block->dma2_array) = 0x0000;
}

return(PASSED);
}

/*****
*
*           END INITIALIZATION ROUTINES
*
*****/

```

```

/*****
*
*          BEGIN SERVICE ROUTINES
*
*****/

```

```

/*****
* ESTAB_DIR () - Establishes the directory structure.
*****/
/
int estab_dir (DATABLOCK *block)
{
if((strcmp(block->wdname, "NULL") == 0)
{
    mkdir ("backs");
    mkdir ("darks");
    mkdir("flats");
    mkdir("images");
}
else
{
    if(chdir (block->wdname)!= FALSE)
    {
        mkdir (block->wdname);
        chdir (block->wdname);
    }
    mkdir ("backs");
    mkdir ("darks");
    mkdir("flats");
    mkdir("images");
}
block->flatctr=0;
block->darkctr=0;
block->backctr=0;
block->imagectr=0;
return(PASSED);
}

```

```

/*****
 *
 *   DMA INTERFACE ROUTINES (lifted from Spectral Systems sample code, as
modified to fit our needs )
 *
 *****/

void  do_dma(DATABLOCK *block)
{
  SetCtrlAttribute(block->mainmenu,MAIN_dma_led,ATTR_OFF_COLOR,VAL_YELLOW);
  if(block->frame1_active == TRUE)
  {
    init_dma(NROWS, NCOLS, block->dma1_array);      /* set up the DMA
array */
    start_dma(0);                                     /* start
DMA in non-video mode */
  }
  if(block->frame2_active == TRUE)
  {
    init_dma(NROWS, NCOLS, block->dma2_array);      /* set up the DMA
array */
    start_dma(0);                                     /* start
DMA in non-video mode */
  }
  return;
}

void quit_dma()
{
  end_dma();
  return;
}

/*****
 *

```

```

*    STUBS
*
*****/

int close_program(DATABLOCK *block)
{
    SetDrive(block->homedrive);
    SetDir (block->homedir);
    remove_isr();
    DiscardPanel(block->mainmenu);
    return (EXIT);
}

int send_osh(DATABLOCK *block)
{
    if(block->outarm == FALSE)
    {
        strcpy(block->outbuffer,"OSH ");
        block->outbuffer[5] = CR;
        block->outarm = TRUE;
        block->outcount = 6;
        block->outptr = block->outbuffer;
        SetCtrlAttribute(block->mainmenu, MAIN_timeout_timer, ATTR_INTERVAL,
(double)1);
        SetCtrlAttribute(block->mainmenu, MAIN_timeout_timer, ATTR_ENABLED,
TRUE);
        return(PASSED);
    }
    return(ERROR);
}

int send_csh(DATABLOCK *block)
{
    if(block->outarm == FALSE)
    {
        strcpy(block->outbuffer,"CSH ");
        block->outbuffer[5] = CR;
        block->outarm = TRUE;
        block->outcount = 6;
    }
}

```

```

        block->outptr = block->outbuffer;
        SetCtrlAttribute(block->mainmenu, MAIN_timeout_timer, ATTR_INTERVAL,
(double)1);
        SetCtrlAttribute(block->mainmenu, MAIN_timeout_timer, ATTR_ENABLED,
TRUE);
        return(PASSED);
    }
    return(ERROR);
}

```

```

int    send_wrm(DATABLOCK *block)
{

    unsigned int    address;
    unsigned int    value;
    int             memsel;
    int             work;

    if(block->outarm == FALSE)
    {
        GetCtrlVal(block->debugmenu,DEBUG_address,&address);
        GetCtrlVal(block->debugmenu,DEBUG_value, &value);
        GetCtrlVal(block->debugmenu,DEBUG_memsel, &memsel);

        strcpy(block->outbuffer,"WRM");
        switch(memsel)
        {
            case    PROGRAM_MEMORY:
            {
                block->outbuffer[3] = 0x10;
                break;
            }
            case    X_MEMORY:
            {
                block->outbuffer[3] = 0x20;
                break;
            }
            case    Y_MEMORY:
            {
                block->outbuffer[3] = 0x40;
            }
        }
    }
}

```

```

        break;
    }
}
block->outbuffer[4] = (address & 0x0000FF00) >> 8;
block->outbuffer[5] = (address & 0x000000FF) ;
block->outbuffer[6] = (value & 0x00FF0000) >> 16;
block->outbuffer[7] = (value & 0x0000FF00) >> 8;
block->outbuffer[8] = (value & 0x000000FF) ;
block->outbuffer[9] = 0x20;
block->outbuffer[10] = 0x20;
block->outbuffer[11] = CR;
block->outarm = TRUE;
block->outcount = 12;
block->outptr = block->outbuffer;
SetCtrlAttribute(block->mainmenu, MAIN_timeout_timer, ATTR_INTERVAL,
(double)1);
SetCtrlAttribute(block->mainmenu, MAIN_timeout_timer, ATTR_ENABLED,
TRUE);
return(PASSED);
}
return(ERROR);
}

```

```

int    send_rdm(DATABLOCK *block)
{

    unsigned int    address;
    int             value;
    int             memsel;
    int             work;

    if(block->outarm == FALSE)
    {
        GetCtrlVal(block->debugmenu, DEBUG_address, &address);
        GetCtrlVal(block->debugmenu, DEBUG_value, &value);
        GetCtrlVal(block->debugmenu, DEBUG_memsel, &memsel);

        block->return_expected = TRUE;
        block->return_ptr.menu = block->debugmenu;
        block->return_ptr.item = DEBUG_value;
    }
}

```

```

    block->return_ptr.addr = &value;

    strcpy(block->outbuffer,"RDM");
    switch(memsel)
    {
        case PROGRAM_MEMORY:
        {
            block->outbuffer[3] = 0x10;
            break;
        }
        case X_MEMORY:
        {
            block->outbuffer[3] = 0x20;
            break;
        }
        case Y_MEMORY:
        {
            block->outbuffer[3] = 0x40;
            break;
        }
    }

    block->outbuffer[4] = (address & 0x0000FF00) >> 8 ;
    block->outbuffer[5] = (address & 0x000000FF) ;
    block->outbuffer[6] = 0x20;
    block->outbuffer[7] = 0x20;
    block->outbuffer[8] = CR;
    block->outarm = TRUE;
    block->outcount = 9;
    block->outptr = block->outbuffer;
    SetCtrlAttribute(block->mainmenu, MAIN_timeout_timer, ATTR_INTERVAL,
(double)1);
    SetCtrlAttribute(block->mainmenu, MAIN_timeout_timer, ATTR_ENABLED,
TRUE);
    return(PASSED);
}
return(ERROR);
}

```

```

int    send_institution(DATABLOCK *block)
{

```

```

unsigned int      address;
int               value;
int               memsel;
int               work;

if(block->outarm == FALSE)
{
    block->return_expected = TRUE;
    block->return_ptr.menu = block->debugmenu;
    block->return_ptr.item = DEBUG_inst_code;
    block->return_ptr.addr = &value;

    strcpy(block->outbuffer,"RDM");
    block->outbuffer[3] = 0x10;          /* reads from program memory only
*/

    block->outbuffer[4] = 0x00;          /* reads from fixed address */
    block->outbuffer[5] = 0x06;
    block->outbuffer[6] = 0x20;
    block->outbuffer[7] = 0x20;
    block->outbuffer[8] = CR;
    block->outarm = TRUE;
    block->outcount = 9;
    block->outptr = block->outbuffer;
    SetCtrlAttribute(block->mainmenu, MAIN_timeout_timer, ATTR_INTERVAL,
(double)1);
    SetCtrlAttribute(block->mainmenu, MAIN_timeout_timer, ATTR_ENABLED,
TRUE);
    return(PASSED);
}
return(ERROR);
}

int  send_firmware(DATABLOCK *block)
{
    unsigned int      address;
    int               value;
    int               memsel;
    int               work;

```

```

if(block->outarm == FALSE)
{
    block->return_expected = TRUE;
    block->return_ptr.menu = block->debugmenu;
    block->return_ptr.item = DEBUG_firmware_code;
    block->return_ptr.addr = &value;

    strcpy(block->outbuffer,"RDM");
    block->outbuffer[3] = 0x10;          /* reads from program memory only
*/

    block->outbuffer[4] = 0x00          /* reads from fixed address */
    block->outbuffer[5] = 0x07
    block->outbuffer[6] = 0x20
    block->outbuffer[7] = 0x20
    block->outbuffer[8] = CR;
    block->outarm = TRUE;
    block->outcount = 9;
    block->outptr = block->outbuffer;
    SetCtrlAttribute(block->mainmenu, MAIN_timeout_timer, ATTR_INTERVAL,
(double)1);
    SetCtrlAttribute(block->mainmenu, MAIN_timeout_timer, ATTR_ENABLED,
TRUE);
    return(PASSED);
}
return(ERROR);
}

```

```

int send_tst(DATABLOCK *block)
{
if(block->outarm == FALSE)
{
    strcpy(block->outbuffer,"TST ");
    block->outbuffer[5] = CR;
    block->outarm=TRUE;
    block->outcount=6;
    block->outptr = block->outbuffer;

```

```

SetCtrlAttribute(block->mainmenu,MAIN_timeout_timer,ATTR_INTERVAL,(double)1)
;
SetCtrlAttribute(block->mainmenu,MAIN_timeout_timer,ATTR_ENABLED,
TRUE);
return(PASSED);
}
return(ERROR);
}

```

```

int send_rrr(DATABLOCK *block)
{
int i_work;
if(block->outarm == FALSE)
{
strcpy(block->outbuffer,"RRR ");
block->outbuffer[5] = CR;
block->outarm=TRUE;
block->outcount=6;
block->outptr = block->outbuffer;
i_work=(block->exp_tim/1000); /* make sure timeout timer
has enough time to */
if(i_work == 0) i_work++; /* do entire exposure, plus
one second */
i_work++;
SetCtrlAttribute(block->mainmenu,MAIN_timeout_timer,ATTR_INTERVAL,(d
ouble)i_work));
SetCtrlAttribute(block->mainmenu,MAIN_timeout_timer,ATTR_ENABLED,
TRUE);
return(PASSED);
}
return(ERROR);
}

```

```

int send_err(DATABLOCK *block)
{
if(block->outarm == FALSE)
{
strcpy(block->outbuffer,"ERR ");
block->outbuffer[5] = CR;
block->outarm=TRUE;
block->outcount=6;
block->outptr = block->outbuffer;

```

```

SetCtrlAttribute(block->mainmenu,MAIN_timeout_timer,ATTR_INTERVAL,(double)1)
;
    SetCtrlAttribute(block->mainmenu,MAIN_timeout_timer,ATTR_ENABLED,
TRUE);
    return(PASSED);
}
return(ERROR);
}

int send_abr(DATABLOCK *block)
{
if(block->outarm == FALSE)
{
    strcpy(block->outbuffer,"ABR ");
    block->outbuffer[5] = CR;
    block->outarm=TRUE;
    block->outcount=6;
    block->outptr = block->outbuffer;

SetCtrlAttribute(block->mainmenu,MAIN_timeout_timer,ATTR_INTERVAL,(double)1)
;
    SetCtrlAttribute(block->mainmenu,MAIN_timeout_timer,ATTR_ENABLED,
TRUE);
    return(PASSED);
}
return(ERROR);
}

int send_sex(DATABLOCK *block)
{
if(block->outarm == FALSE)
{
    strcpy(block->outbuffer,"SEX");
    block->outbuffer[3] = ((block->exp_tim & 0x00FF0000) >> 16);
    block->outbuffer[4] = ((block->exp_tim & 0x0000FF00) >> 8);
    block->outbuffer[5] = ((block->exp_tim & 0x000000FF) );
    block->outbuffer[6] = 0x20;
    block->outbuffer[7] = 0x20;
    block->outbuffer[8] = CR;
    block->outarm=TRUE;
    block->outcount=9;
    block->outptr = block->outbuffer;

SetCtrlAttribute(block->mainmenu,MAIN_timeout_timer,ATTR_INTERVAL,(double)1)

```

```

;
    SetCtrlAttribute(block->mainmenu,MAIN_timeout_timer,ATTR_ENABLED,
TRUE);
    return(PASSED);
}
return(ERROR);
}

int send_ldw(DATABLOCK *block)
{
if(block->outarm == FALSE)
{
    strcpy(block->outbuffer,"LDW ");
    block->outbuffer[3] = ((block->port_flags & 0x00FF0000) >> 16);
    block->outbuffer[4] = ((block->port_flags & 0x0000FF00) >> 8);
    block->outbuffer[5] = ((block->port_flags & 0x000000FF) );
    block->outbuffer[8] = CR;
    block->outarm=TRUE;
    block->outcount=9;
    block->outptr = block->outbuffer;

SetCtrlAttribute(block->mainmenu,MAIN_timeout_timer,ATTR_INTERVAL,(double)1)
;
    SetCtrlAttribute(block->mainmenu,MAIN_timeout_timer,ATTR_ENABLED,
TRUE);
    return(PASSED);
}
return(ERROR);
}
int send_sb2(DATABLOCK *block)
{
if(block->outarm == FALSE)
{
    strcpy(block->outbuffer,"SB2 ");
    block->outbuffer[3] = ((block->bias & 0x00FF0000) >> 16);
    block->outbuffer[4] = ((block->bias & 0x0000FF00) >> 8);
    block->outbuffer[5] = ((block->bias & 0x000000FF) );
    block->outbuffer[6] = 0x20;
    block->outbuffer[7] = 0x20;
    block->outbuffer[8] = CR;
    block->outarm=TRUE;
    block->outcount=9;
    block->outptr = block->outbuffer;

```

```

SetCtrlAttribute(block->mainmenu,MAIN_timeout_timer,ATTR_INTERVAL,(double)1)
;
SetCtrlAttribute(block->mainmenu,MAIN_timeout_timer,ATTR_ENABLED,
TRUE);
return(PASSED);
}
return(ERROR);
}

```

```

int send_sgn(DATABLOCK *block)
{
if(block->outarm == FALSE)
{
strcpy(block->outbuffer,"SGN ");
block->outbuffer[3] = ((block->gain & 0x00FF0000) >> 16);
block->outbuffer[4] = ((block->gain & 0x0000FF00) >> 8);
block->outbuffer[5] = ((block->gain & 0x000000FF) );
block->outbuffer[6] = 0x20;
block->outbuffer[7] = 0x20;
block->outbuffer[8] = CR;
block->outarm=TRUE;
block->outcount=9;
block->outptr = block->outbuffer;

```

```

SetCtrlAttribute(block->mainmenu,MAIN_timeout_timer,ATTR_INTERVAL,(double)1)
;
SetCtrlAttribute(block->mainmenu,MAIN_timeout_timer,ATTR_ENABLED,
TRUE);
return(PASSED);
}
return(ERROR);
}

```

```

int send_rst(DATABLOCK *block)
{
if(block->outarm == FALSE)
{
strcpy(block->outbuffer,"RST ");
block->outbuffer[5] = CR;
block->outarm=TRUE;

```

```

        block->outcount=6;
        block->outptr = block->outbuffer;

        SetCtrlAttribute(block->mainmenu,MAIN_timeout_timer,ATTR_INTERVAL,(double)1)
        ;
        SetCtrlAttribute(block->mainmenu,MAIN_timeout_timer,ATTR_ENABLED,
        TRUE);
        return(PASSED);
    }
    return(ERROR);
}

int send_pon(DATABLOCK *block)
{
    if(block->outarm == FALSE)
    {
        strcpy(block->outbuffer,"PON ");
        block->outbuffer[5] = CR;
        block->outarm=TRUE;
        block->outcount=6;
        block->outptr = block->outbuffer;

        SetCtrlAttribute(block->mainmenu,MAIN_timeout_timer,ATTR_INTERVAL,(double)1)
        ;
        SetCtrlAttribute(block->mainmenu,MAIN_timeout_timer,ATTR_ENABLED,
        TRUE);
        return(PASSED);
    }
    return(ERROR);
}

int send_pof(DATABLOCK *block)
{
    if(block->outarm == FALSE)
    {
        strcpy(block->outbuffer,"POF ");
        block->outbuffer[5] = CR;
        block->outarm=TRUE;
        block->outcount=6;
        block->outptr = block->outbuffer;
    }
}

```

```

SetCtrlAttribute(block->mainmenu,MAIN_timeout_timer,ATTR_INTERVAL,(double)1)
;
    SetCtrlAttribute(block->mainmenu,MAIN_timeout_timer,ATTR_ENABLED,
TRUE);
    return(PASSED);
}
return(ERROR);
}

```

```

int send_con(DATABLOCK *block)
{
if(block->outarm == FALSE)
{
    strcpy(block->outbuffer,"CON ");
    block->outbuffer[5] = CR;
    block->outarm=TRUE;
    block->outcount=6;
    block->outptr = block->outbuffer;

```

```

SetCtrlAttribute(block->mainmenu,MAIN_timeout_timer,ATTR_INTERVAL,(double)1)
;
    SetCtrlAttribute(block->mainmenu,MAIN_timeout_timer,ATTR_ENABLED,
TRUE);
    return(PASSED);
}
return(ERROR);
}

```

```

int send_cof(DATABLOCK *block)
{
if(block->outarm == FALSE)
{
    strcpy(block->outbuffer,"COF ");
    block->outbuffer[5] = CR;
    block->outarm=TRUE;
    block->outcount=6;
    block->outptr = block->outbuffer;

```

```

SetCtrlAttribute(block->mainmenu,MAIN_timeout_timer,ATTR_INTERVAL,(double)1)
;
    SetCtrlAttribute(block->mainmenu,MAIN_timeout_timer,ATTR_ENABLED,

```

```

TRUE);
    return(PASSED);
}
return(ERROR);
}

```

```

/*****
 *
 *   FITS ROUTINES
 *
 *****/

```

```

int    MakeFits(DATABLOCK *block)
{

char    workstr[80];
        /* code lifted from sample program */
char    workstr2[80];
char    workstr3[80];


int      work;
int      m_icode;
fitsfile *fptr;
        /* pointer to the FITS file */
int      status;
        /* other FITS file stuff */

long     fpixel;
long     naxis;
long     nelements;
long     naxes[2] = {NROWS, NCOLS};
        /* this locks the size to that of the array */
char     cs_errstr[80];


status = FALSE;
fpixel = 1;
naxis = 2;

```

```

if(strlen(block->wddrive)>0)
    /* change drive, ensure file is made in directory */
    {
        if(block->wddrive[0]>0x60)
            /* they don't make it easy */
            {
                block->wddrive[0] -= 0x60;
            }
        else
            {
                block->wddrive[0] -= 0x40;
            }
        SetDrive((int)(block->wddrive[0]));
    }
strcpy(workstr,block->wddir);

GetCtrlVal(block->imagemenu,IMAGE_saveselect,&work);
switch(work)
{
    case 1:          /* flat */
    {
        strcpy(workstr,"flats\\");
        sprintf(workstr2,"FLAT%04d.FTS",block->flatctr);
        block->flatctr++;
        SetCtrlVal(block->imagemenu,IMAGE_flatctr,block->flatctr);
        break;
    }
    case 2:          /* dark */
    {
        strcpy(workstr,"darks\\");
        sprintf(workstr2,"DARK%04d.FTS",block->darkctr);
        block->darkctr++;
        SetCtrlVal(block->imagemenu,IMAGE_darkctr,block->darkctr);
        break;
    }
    case 3:          /* back */
    {
        strcpy(workstr,"backs\\");
        sprintf(workstr2,"BACK%04d.FTS",block->backctr);
        block->backctr++;
        SetCtrlVal(block->imagemenu,IMAGE_backctr,block->backctr);
        break;
    }
}

```

```

        case 4:          /* image */
        {
            strcpy(workstr,"images\\");
            sprintf(workstr2,"IMAGE%03d.FTS",block->imagectr);
            block->imagectr++;
            SetCtrlVal(block->imagemenu,IMAGE_imagectr,block->imagectr);
            break;
        }
    }
    strcpy(workstr3,block->wddir);
    strcat(workstr3,workstr);
    m_ecode = SetDir(workstr3);

    m_ecode = FITS_create_file(&fptr,workstr2,&status);
    /* make the FITS file */
    if(m_ecode!=0)
    {
        sprintf(cs_errstr,"FITS FILE CREATION ERROR, CODE IS %d",status);
        error_message(block,cs_errstr);
        return(status);
    }

    m_ecode = FITS_create_img(fptr, USHORT_IMG, naxis, naxes, &status);
    if(m_ecode!=0)
    {
        sprintf(cs_errstr,"FITS IMAGE CREATION ERROR, CODE IS %d",status);
        error_message(block,cs_errstr);
        return(status);
    }

    fpixel = 1;
    nelements = NROWS * NCOLS;

    /* now, write the image. */

    if(block->frame1_active == TRUE) m_ecode = FITS_write_img(fptr, TUSHORT, fpixel,
nelements, block->dma1_array, &status);

```

```

if(block->frame2_active == TRUE) m_ecode = FITS_write_img(fp, TUSHORT, fpixel,
nelements, block->dma2_array, &status);
if(m_ecode!=0)
{
    sprintf(cs_errstr,"FITS WRITE IMAGE ERROR, CODE IS %d",status);
    error_message(block,cs_errstr);
    return(status);
}

/* keywords must be written AFTER the image is loaded, not before.
* the reason is because the first keyword must be "SIMPLE", and
* if it is not then the FITS package gets upset and goes off into the night.
*/

/* first, write the date/time group */
if(block->frame1_active == TRUE) m_ecode = FITS_update_key(fp,TSTRING,"DATE
",NULL, block->timestamp1,&status);
if(block->frame2_active == TRUE) m_ecode = FITS_update_key(fp,TSTRING,"DATE
",NULL, block->timestamp2,&status);

m_ecode = GetCtrlVal(block->setupmenu,SETUP_origin,workstr);
/* origin */
m_ecode = FITS_update_key(fp,TSTRING,"ORIGIN ",NULL,workstr,&status);

m_ecode = GetCtrlVal(block->setupmenu,SETUP_telescop,workstr);
/* telescope */
m_ecode = FITS_update_key(fp,TSTRING,"TELESCOP",NULL,workstr,&status);

m_ecode = GetCtrlVal(block->setupmenu,SETUP_instrume,workstr);
/* instrument */
m_ecode = FITS_update_key(fp,TSTRING,"INSTRUME",NULL,workstr,&status);

m_ecode = GetCtrlVal(block->setupmenu,SETUP_observer,workstr);
/* observer */
m_ecode = FITS_update_key(fp,TSTRING,"OBSERVER",NULL,workstr,&status);

m_ecode = GetCtrlVal(block->imagemenu,IMAGE_object,workstr);
/* object */
m_ecode = FITS_update_key(fp,TSTRING,"OBJECT ",NULL,workstr,&status);

m_ecode = GetCtrlVal(block->setupmenu,SETUP_equinox,workstr);

```

```

/* equinox */
m_icode = FITS_update_key(fp, TSTRING, "EQUINOX ", NULL, workstr, &status);

m_icode = GetCtrlVal(block->setupmenu, SETUP_comment, workstr);
/* comment */
m_icode = FITS_update_key(fp, TSTRING, "COMMENT ", NULL, workstr, &status);

m_icode = GetCtrlVal(block->setupmenu, SETUP_timesys, workstr);
/* timesys */
m_icode = FITS_update_key(fp, TSTRING, "TIMESYS ", NULL, workstr, &status);

m_icode = GetCtrlVal(block->imagemenu, IMAGE_ra, workstr);
/* ra */
m_icode = FITS_update_key(fp, TSTRING, "RA   ", NULL, workstr, &status);

m_icode = GetCtrlVal(block->imagemenu, IMAGE_dec, workstr);
/* dec */
m_icode = FITS_update_key(fp, TSTRING, "DEC   ", NULL, workstr, &status);

m_icode = GetCtrlVal(block->imagemenu, IMAGE_exp_tim, &work);
/* exposure time */
sprintf(workstr, "%d MILLISECONDS", work);
m_icode = FITS_update_key(fp, TSTRING, "EXPOSURE", NULL, workstr, &status);

sprintf(workstr, "%d FILTER", block->active_filter_number);
/* filter */
m_icode = FITS_update_key(fp, TSTRING, "FILTER ", NULL, workstr, &status);

m_icode = GetCtrlVal(block->imagemenu, IMAGE_air_mass, workstr);
/* air mass */
m_icode = FITS_update_key(fp, TSTRING, "AIR MASS", NULL, workstr, &status);

/* close the FITS file and report any errors */
FITS_close_file(fp, &status);
FITS_report_error(stderr, status);

/* the very last thing we need to do before we exit is to reset the working directory. */
/* must use full pathname, not just a relative path */
SetDir(block->wdname);

return(status);
}

```

```

/*****
*
*   END FITS HANDLERS
*
*****/

```

```

/*****
*
*   CONFIG FILE HANDLING ROUTINES
*
*****/

```

```

/* the reason this is split out is because the config file
* may not necessarily be IN the startup directory.
*/

```

```

int    get_home_dir(DATABLOCK *block)
{
int      m_icode;
int      work;
char    workstr [80];

/* get home drive and directory */
m_icode = GetDrive(&(block->homedrive), &work);
if(m_icode < (-1))
{
    sprintf(workstr, "CONFIG READ: I/O ERROR ON GET LOCAL DRIVE, CODE
%d", m_icode);
    error_message(block, workstr);
    return(FAILCODE);
}
m_icode = GetDir(block->homedir);
if(m_icode < 0)
{
    sprintf(workstr, "CONFIG READ: I/O ERROR ON GET LOCAL DIR, CODE
%d", m_icode);

```

```

        error_message(block,workstr);
        return(FAILCODE);
    }
    return(SUCCESS);
}

```

```

int    config_read(DATABLOCK *block)
{
    FILE          *fsconfig_file;
    int            m_icode;
    int            work;
    char           workstr[80];
    char           *cp_ptr;
    int            filesize;
    int            i_work;
    char           cs_workstr2[80];
    unsigned       ui_work;

```

```

    /*try to open the configuration file */
    GetCtrlVal(block->filemenu,FILEMENU_config_file,workstr);
    m_icode=GetFileInfo(workstr,&filesize);
    if(m_icode < (1))
    {
        sprintf(workstr, "CONFIG READ: CONFIG FILE ERROR CODE
%d",m_icode);
        error_message(block,workstr);
        return(FAILCODE);
    }
    fsconfig_file=fopen(workstr,"r");

```

```

    /* if we get this far, read the file and save the contents */
    /* NOTE: This routine contains the cracker
    * it doesn't generate errors, it just ignores strings it can't identify
    */
    while(fgets(workstr,CARD_SIZE,fsconfig_file) != NULL)
    {
        if(strstr(workstr,"FILTERPORT=") != NULL)

            {
                cp_ptr= strchr(workstr,'=');
                cp_ptr++;

```

```

        sscanf(cp_ptr,"%d",&block->port_base);
        OutPort(block->port_base+PORT_D_OFFSET,
PORT_CONTROL_MASK);      /* a little something is to initialize the */
    }

                                /* 8255 port control register */
    if(strstr(workstr,"UTC_OFFSET=")!=NULL)
    {
        cp_ptr=strchr(workstr,'=');
        cp_ptr++;
        sscanf(cp_ptr,"%d",&i_work);
        SetCtrlVal(block->imagemenu,IMAGE_utc_offset,i_work);
    }
    if(strstr(workstr,"RA=")!=NULL)
    {
        cp_ptr=strchr(workstr,'=');
        cp_ptr++;
        strcpy(cs_workstr2,cp_ptr);
        SetCtrlVal(block->imagemenu,IMAGE_ra,cs_workstr2);
    }
    if(strstr(workstr,"DEC=")!=NULL)
    {
        cp_ptr=strchr(workstr,'=');
        cp_ptr++;
        strcpy(cs_workstr2,cp_ptr);
        SetCtrlVal(block->imagemenu,IMAGE_dec,cs_workstr2);
    }
    if(strstr(workstr,"AIR_MASS=")!=NULL)
    {
        cp_ptr=strchr(workstr,'=');
        cp_ptr++;
        strcpy(cs_workstr2,cp_ptr);
        SetCtrlVal(block->imagemenu,IMAGE_air_mass,cs_workstr2);
    }
    if(strstr(workstr,"OBJECT=")!=NULL)
    {
        cp_ptr=strchr(workstr,'=');
        cp_ptr++;
        strcpy(cs_workstr2,cp_ptr);
        SetCtrlVal(block->imagemenu,IMAGE_object,cs_workstr2);
    }
    if(strstr(workstr,"EXP_TIME=")!=NULL)
    {
        cp_ptr=strchr(workstr,'=');

```

```

        cp_ptr++;
        sscanf(cp_ptr,"%d",&ui_work);
        SetCtrlVal(block->imagemenu,IMAGE_exp_tim,ui_work);
        SetCtrlVal(block->debugmenu,DEBUG_exp_tim,ui_work);
    }
    if(strstr(workstr,"BIAS=")!=NULL)
    {
        cp_ptr=strchr(workstr,'=');
        cp_ptr++;
        sscanf(cp_ptr,"%d",&i_work);
        SetCtrlVal(block->imagemenu,IMAGE_bias,i_work);
    }
    if(strstr(workstr,"GAIN=")!=NULL)
    {
        cp_ptr=strchr(workstr,'=');
        cp_ptr++;
        sscanf(cp_ptr,"%d",&ui_work);
        SetCtrlVal(block->imagemenu,IMAGE_gain,ui_work);
    }

    if(strstr(workstr,"ORIGIN=")!=NULL)
    {
        cp_ptr=strchr(workstr,'=');
        cp_ptr++;
        strcpy(cs_workstr2,cp_ptr);
        SetCtrlVal(block->setupmenu,SETUP_origin,cs_workstr2);
    }
    if(strstr(workstr,"TELESCOP=")!=NULL)
    {
        cp_ptr=strchr(workstr,'=');
        cp_ptr++;
        strcpy(cs_workstr2,cp_ptr);
        SetCtrlVal(block->setupmenu,SETUP_telescop,cs_workstr2);
    }
    if(strstr(workstr,"INSTRUME=")!=NULL)
    {
        cp_ptr=strchr(workstr,'=');
        cp_ptr++;
        strcpy(cs_workstr2,cp_ptr);
        SetCtrlVal(block->setupmenu,SETUP_instrume,cs_workstr2);
    }
    if(strstr(workstr,"OBSERVER=")!=NULL)
    {

```

```

        cp_ptr=strchr(workstr,'=');
        cp_ptr++;
        strcpy(cs_workstr2,cp_ptr);
        SetCtrlVal(block->setupmenu,SETUP_observer,cs_workstr2);
    }
    if(strstr(workstr,"COMMENT=")!=NULL)
    {
        cp_ptr=strchr(workstr,'=');
        cp_ptr++;
        strcpy(cs_workstr2,cp_ptr);
        SetCtrlVal(block->setupmenu,SETUP_comment,cs_workstr2);
    }
    if(strstr(workstr,"EQUINOX=")!=NULL)
    {
        cp_ptr=strchr(workstr,'=');
        cp_ptr++;
        strcpy(cs_workstr2,cp_ptr);
        SetCtrlVal(block->setupmenu,SETUP_equinox,cs_workstr2);
    }
    if(strstr(workstr,"TIMESYS=")!=NULL)
    {
        cp_ptr=strchr(workstr,'=');
        cp_ptr++;
        strcpy(cs_workstr2,cp_ptr);
        SetCtrlVal(block->setupmenu,SETUP_timesys,cs_workstr2);
    }
    if(strstr(workstr,"TPFILE=")!=NULL)
    {
        cp_ptr=strchr(workstr,'=');
        cp_ptr++;
        strcpy(cs_workstr2,cp_ptr);
        SetCtrlVal(block->setupmenu,SETUP_tpfile,cs_workstr2);
    }
}

/* now, echo these back to the various screens */

SetCtrlVal(block->setupmenu,SETUP_port_base, block->port_base);

/* then close the file and return */
fclose(fsconfig_file);

```

```

return(SUCCESS);
}

/* write the configuration file out into whatever directory is active at the time */
int config_write(DATABLOCK *block)
{
    FILE                *fsp_config_file;
    char                cs_workstr [80];
    unsigned int        ui_work;
    unsigned short int  usi_work;
    int                 i_work;

    /* get the name of the config file and open it */
    GetCtrlVal(block->filemenu, FILEMENU_config_file, cs_workstr);
    fsp_config_file = fopen(cs_workstr, "w");

    /* now, dump the lot */
    GetCtrlVal(block->setupmenu, SETUP_port_base, &usi_work);
    fprintf(fsp_config_file, "FILTERPORT=%d\n", usi_work);

    GetCtrlVal(block->setupmenu, SETUP_origin, cs_workstr);
    fprintf(fsp_config_file, "ORIGIN=%s\n", cs_workstr);

    GetCtrlVal(block->setupmenu, SETUP_telescop, cs_workstr);
    fprintf(fsp_config_file, "TELESCOP=%s\n", cs_workstr);

    GetCtrlVal(block->setupmenu, SETUP_instrume, cs_workstr);
    fprintf(fsp_config_file, "INSTRUME=%s\n", cs_workstr);

    GetCtrlVal(block->setupmenu, SETUP_observer, cs_workstr);
    fprintf(fsp_config_file, "OBSERVER=%s\n", cs_workstr);

    GetCtrlVal(block->setupmenu, SETUP_comment, cs_workstr);
    fprintf(fsp_config_file, "COMMENT=%s\n", cs_workstr);

    GetCtrlVal(block->setupmenu, SETUP_timesys, cs_workstr);
    fprintf(fsp_config_file, "TIMESYS=%s\n", cs_workstr);

    GetCtrlVal(block->setupmenu, SETUP_equinox, cs_workstr);
    fprintf(fsp_config_file, "EQUINOX=%s\n", cs_workstr);

```

```

GetCtrlVal(block->setupmenu, SETUP_tpfile,cs_workstr);
fprintf(fsp_config_file,"TPFILE=%s\n",cs_workstr);


GetCtrlVal(block->imagemenu, IMAGE_utc_offset,&i_work);
fprintf(fsp_config_file,"UTC_OFFSET=%d\n",i_work);


GetCtrlVal(block->imagemenu, IMAGE_ra,cs_workstr);
fprintf(fsp_config_file,"RA=%s\n",cs_workstr);


GetCtrlVal(block->imagemenu, IMAGE_dec,cs_workstr);
fprintf(fsp_config_file,"DEC=%s\n",cs_workstr);


GetCtrlVal(block->imagemenu, IMAGE_air_mass,cs_workstr);
fprintf(fsp_config_file,"AIR_MASS=%s\n",cs_workstr);


GetCtrlVal(block->imagemenu, IMAGE_object,cs_workstr);
fprintf(fsp_config_file,"OBJECT=%s\n",cs_workstr);


GetCtrlVal(block->imagemenu, IMAGE_exp_tim,&ui_work);
fprintf(fsp_config_file,"EXP_TIME=%d\n",ui_work);


GetCtrlVal(block->imagemenu, IMAGE_bias,&i_work);
fprintf(fsp_config_file,"BIAS=%d\n",i_work);


GetCtrlVal(block->imagemenu, IMAGE_gain,&i_work);
fprintf(fsp_config_file,"GAIN=%d\n",i_work);


/* close the config file and exit */
fclose(fsp_config_file);
return(SUCCESS);
}

```

D.1.5 Listing of xlate.c

```

#include <ansi_c.h>
#include <utility.h>
#include <stdio.h>

#define _CRTBLD                                     // what a royal pain it is, trying to
#define _INTERNAL_IFSTRIP_                         // keep everyone happy...
#define _WCHAR_T_DEFINED
#include "msctype.h"
#undef      _WCHAR_T_DEFINED
#undef      _INTERNAL_IFSTRIP_
#undef      _CRTBLD

FILE _iob[20];

/* a global for the FITS library */
int  _errno;
int  __mb_cur_max = 1;

/* these map ANSI functions to CVI functions */

char* _getdcwd(int drive, char* buffer, int maxlen)
{
    if(drive != NULL)
    {
        SetDrive(drive);
    }
    GetDir(buffer);
    return(buffer);
}

int  _isctype(int c, int mask)
{
    if(((unsigned)(c+1))<=256)
    {
        return(_pctype[c] & mask);
    }
    else

```

```
    {
        return(0);
    }
}

int chdir(char *instring)
{
    return(SetDir(instring));
}

int chdrive(int driveno)
{
    return(SetDrive(driveno));
}

int pwd(char *instring)
{
    return(GetDir(instring));
}

int mkdir(char *instring)
{
    return(MakeDir(instring));
}

/* these map MSVC5 functions to CVI functions */
float _CItanh(float arg)
{
    return(tanh (arg));
}

float _CIsinh(float arg)
{
    return(sinh(arg));
}

float _CIacos(float arg)
{
    return(acos(arg));
}

float _CIasin(float arg)
{

```

```
return(asin(arg));  
}
```

```
float _Cipow(float base,float exponent)  
{  
return(pow(base,exponent));  
}
```

```
float _Cicosh(float arg)  
{  
return(cosh(arg));  
}
```

D.1.6 Listing of xlate.h

```
int mkdir (char *);  
int chdir (char *);  
int pwd  (char *);  
int setdrive(int);
```

D.1.7 Listing of globa.l.h

```

/*****
 *
 * GLOBAL.H - This is the global define include file.
 *
 * $Id: GLOBAL.H 1.1 1996/03/27 KRS Exp $
 * $Id: GLOBAL.H 2.01 2001/05/17 SEJ Exp &
 *****/
/
#include <stdio.h>
#include <stdlib.h>
/*****
 * DEFINES
 *****/
/

typedef unsigned char byte; /* 8-bit */
typedef unsigned short word; /* 16-bit */
typedef unsigned int dword; /* 32-bit */
typedef unsigned long qword; /* 64-bit */

```

D.1.8

```

/*****
* SVID.H
* This is the CVI imaging header file specifically designed
* for the IRL MCE-3 NICMOS infrared camera and the Spectral
* Instruments DIO board.
*
* Not anymore it isn't. AJ
*
*****/

#define BYTES_PER_PIX 2
#define NROWS 128
#define NCOLS 128

#define ARRAY_SIZE NROWS * NCOLS
#define ARRAY_DIM NROWS, NCOLS
#define ARRAY_BYTES ARRAY_SIZE * BYTES_PER_PIX

/***** ERROR CODE DEFINES *****/
#define PASSED 0x0000
#define DMA_ARRAY1_ALLOCATION_FAILURE 0x0100
#define DMA_ARRAY2_ALLOCATION_FAILURE 0x0101

/*****
* GLOBALS -
*****/

#define ABORT 666 /* appropriate for an abort

```

```

flag */
#define EXIT 1728          /* RIP LVB */
#define FAILCODE -1        /* generic fail and success codes */
#define SUCCESS 0

/* definitions for the serial port on the Spectral Instruments card */
#define BAUD 9600
#define PARITY 0
#define BITS 8
#define STOP 1
#define BUFFERSIZE 0x2000
#define CR 0x0D
#define BACKSLASH 0x5C

#define Winhndl int
#define Menhndl int
#define WORD unsigned short int
#define DWORD unsigned int
#define BYTE unsigned char
#define CHAR char
#define LPSTR void *
#define UCHAR unsigned char

/* the next three must have this value to match the memory select toggle in
 * the debug screen */
#define PROGRAM_MEMORY 1
#define X_MEMORY 2
#define Y_MEMORY 3

/* these are for the DMA transfer routines. These values match the returns from
dma_done() */
#define DMA_TRANSFER_IN_PROGRESS 0
#define DMA_COMPLETE 1
#define DMA_ERROR 2

#define INDIAN_FILE_SIZE 12288 /* number of bytes in
indian head test pattern */

typedef struct
{
    Winhndl menu;
    int item;

```

```

int          *addr;
} RTN_PTR;

typedef struct                                /* this typedef right here is the data object
used throughout the application */
{
    /* stuff to handle returns from the controller */
    RTN_PTR    return_ptr;
    BOOL return_expected;

    /* menu control stuff */
    char  instr[80];          /* incoming string */
    char  outstr[80];         /* outgoing string */
    char  fname[80];          /* file name */
    char  workstr[80];        /* work string */
    char  message[80];        /* message passed to message screen */
    Winhndl  mainmenu;        /* main screen handle */
    Winhndl  debugmenu;       /* debug screen handle */
    Winhndl  filemenu;        /* file screen handle */
    Winhndl  errormenu;       /* error message screen handle */
    Winhndl  imagemenu;       /* image display, fetch, and storage cntl
handle */
    Winhndl  setupmenu;       /* mostly string entry for FITS file
writing */
    Winhndl  filtermenu;      /* filter wheel control menu */
    Winhndl  aboutmenu;       /* about control menu */
    Menhndl  menuhandle;      /* menu bar handle for main menu */

    /* working directory stuff */
    char  wdname[80];         /* working directory
full path */
    char  wddrive[MAX_DRIVENAME_LEN]; /* drive name */
    char  wddir[MAX_DIRNAME_LEN];    /* directory name */
    char  wdfile[MAX_FILENAME_LEN];   /* file name (should be null)
*/
    BOOL fileset_open;        /* marks if fileset has been
initialized */

    /* home directory stuff */
    int    homedrive;         /* home drive
*/

```

```

char    homedir    [512];                /* home directory */

/* dma stuff */
WORD    *dma1_array;                    /* handles to the dma
blocks */
WORD    *dma2_array;
BOOL dma_running;                        /* DMA running flag */
BOOL frame1_active;                    /* marks which frame
is active */
BOOL frame2_active;
CHAR timestamp1    [80];                /* timestamps for the data for
FITS storage */
CHAR timestamp2    [80];
CHAR timehack      [80];                /* where the raw time hack
goes. Updated every second */
int        oldseconds;                    /* updated
once per second */

/* variables, flags, etc. */
DWORD    exp_tim;                        /* exposure time, milliseconds */
DWORD    port_flags;                    /* controller port bitmapped flags,
for debugging and test */
int        bias;                        /* ADC bias */
int        gain;                        /* ADC board gain */

/* serial port stuff */
UCHAR    outbuffer [80];                /* outbound buffer */
int        outcount;                    /* count of outgoing chars */
UCHAR * outptr;                        /* pointer to current outgoing char */
BOOL outarm;                            /* arming toggle */

UCHAR    inbuffer [80];                /* inbound buffer */
int        incount;                    /* count of number of chars in
incoming buffer */
UCHAR * inptr;                        /* pointer to end char in buffer */
BOOL inarm;                            /* arming toggle */

int        flatctr;                    /* flat sequence counter */
int        darkctr;                    /* dark sequence counter */
int        backctr;                    /* back sequence counter */
int        imagectr;                    /* image sequence counter */

```

```

Point frame1point;      /* point structs as used in CVI */
Point frame2point;

/* debug stuff */
BOOL whiterat;          /* the white rat toggle */

/* filter handler stuff */
BOOL home_found;        /* marks if fiducial is found */
BOOL filter_moving;     /* marks if filter wheel is moving */
int  active_filter_number; /* the number of the filter in front of the detector */
int  active_filter_target; /* the number of the filter to be moved to */
BOOL filter_power;      /* marks whether power is available to the
filter wheel */

/* 8255 port stuff */
unsigned long port_base; /* the 8255 base address */
unsigned short port_a;   /* port A image, 8255 */
unsigned short port_b;   /* port B image, 8255 */
unsigned short port_c;   /* port C image, 8255 */
unsigned short port_cntl; /* control port image, 8255 */

} DATABLOCK;

/** INITIALIZATION FUNCTIONS **/
int init_windows      (DATABLOCK *);
int init_spectral     (DATABLOCK *);
int allocate_buffers  (DATABLOCK *);
int init_object       (DATABLOCK *);

/** SPECTRAL SYSTEMS SERIAL PORT
FUNCTIONS (HIGH LEVEL) */
int receive_serial    (DATABLOCK *);
int send_serial       (DATABLOCK *);
int crack_incoming    (DATABLOCK *);

/** CALLBACKS AND CRACKERS **/
int main_cracker      (int, DATABLOCK *);
int debug_cracker     (int, DATABLOCK *);
int file_cracker      (int, DATABLOCK *);
int error_cracker     (int, DATABLOCK *);
void error_message    (DATABLOCK *, char*); /* service routine, not a cracker.

```

included here because it is an anomaly.

```

int image_cracker  (int, DATABLOCK *);
int  setup_cracker (int, DATABLOCK *);
int filter_cracker  (int, DATABLOCK *);
int run_utc_clock   (DATABLOCK *);
int  about_cracker (int, DATABLOCK *);

    /** PROGRAM CONTROL FUNCTIONS **/
int close_program      (DATABLOCK *);
int config_write       (DATABLOCK *);
int config_read        (DATABLOCK *);
int estab_dir          (DATABLOCK *);

    /** FILTER WHEEL CONTROLS **/
int      run_filterwheel      (DATABLOCK *);
void DeassertFilterPower      (DATABLOCK *);
void AssertFilterPower        (DATABLOCK *);
void ClearFilterIndicators    (DATABLOCK *);
void SetActiveFilterIndicator(DATABLOCK *);

    /** DMA ROUTINES */
int      run_dma              (DATABLOCK *);
void do_dma                    (DATABLOCK *);

    /** FILE ROUTINES */
int      MakeFits              (DATABLOCK *);
int      get_home_dir (DATABLOCK *);

    /** SERVICE ROUTINES */
int      send_tst              (DATABLOCK *);          /* test
message */
int      send_err              (DATABLOCK *);          /* error
message */
int      send_sex              (DATABLOCK *);          /* set
exposure time */
int      send_ldw              (DATABLOCK *);          /* load
parallel word */
int      send_pon              (DATABLOCK *);          /*

```

```

power on */
int      send_rst      (DATABLOCK *);      /* reset
*/
int      send_pof      (DATABLOCK *);      /*
power off */
int      send_osh      (DATABLOCK *);      /* open
shutter */
int      send_csh      (DATABLOCK *);      /*
close shutter */
int      send_wrm      (DATABLOCK *);      /*
write memory */
int      send_rdm      (DATABLOCK *);      /* read
memory */
int      send_rrr      (DATABLOCK *);      /* do
timed exposure with DMA */
int      send_sb2      (DATABLOCK *);      /* send
ADC bias voltage adjust */
int      send_sgn      (DATABLOCK *);      /* send
ADC gain adjust */
int      send_abr      (DATABLOCK *);      /* send
abort */
int      send_con      (DATABLOCK *);      /* send
camera power on */
int      send_cof      (DATABLOCK *);      /* send
camera power off */
int      send_firmware (DATABLOCK *);      /* get
controller firmware version */
int      send_institution(DATABLOCK *);    /* get
controller firmware institution code */

```

```

/** ENDIT **/

```

D.2.1 Bootstrap.asm as supplied by IR Labs

COMMENT *

This file is used to generate boot DSP code for the second generation
TIMII timing board with the PC interface for IR Labs.

This is Rev. 3.00 software.

Overlays are no longer used, but application programs can be loaded.

Modified starting for downloading operation with timIappl.asm

Header ID code eliminated since the utility board will not be used -
may be re-implemented if needed. (Aug. 23, 1996)

Buffers for commands and replies was simplified to just two buffer, one
for the receiver, one for the transmitter. Each has an address register
pointing to the current value of the last entry in the buffer (R1 for
receiver, R3 for transmitter) and an address register pointing to the
last processed entry (R2 for the receiver, R4 for the transmitter).
(Aug. 25, 1996)

SCI interrupt service routine to place the first character in the incoming
stream into the most significant byte of the 3-byte DSP word.
(Aug. 26, 1996)

Timer code based on DSP timer interrupt service added Aug. 31, 1996. It was
verified to work by testing the X:TCSR bit 0 = TE for timer complete.

Modified for Rev. 3 PCI timing boards March '97

Modified for Rev. 6C power board Aug. '98

Base copy 010705 University of Calgary. Code is proved to work.
Designated institution code 010101

*

PAGE 132 ; Printronix page width - 132 columns

; Define some useful DSP register locations

RST_ISR EQU \$00 ; Hardware reset interrupt

ROM_ID EQU \$06 ; Location of program Identification = SWI interrupt

SCI_ISR EQU \$14 ; SCI serial receiver interrupt address

SCI_ERR EQU \$16 ; SCI interrupt with exception (error)

PGM_STR EQU \$18 ; Starting address of program

TIM_ISR EQU \$3C ; DSP timer interrupt service routine address

```

PGM_CON EQU    $3E    ; Program continues on here
BUF_STR EQU    $60    ; Starting address of buffers in X:
BUF_LEN EQU    $20    ; Length of each buffer
RCV_BUF EQU    BUF_STR ; Starting address of serial receiver buffer in X:
XMT_BUF EQU    BUF_STR+BUF_LEN ; Starting address of command buffer in X:
COM_TBL EQU    XMT_BUF+BUF_LEN ; Starting address of command table in X:
NUM_COM EQU    24     ; Number of entries in command table

ROM_OFF EQU    $4000  ; Boot program offset address in EEPROM
LD_X EQU    $4200  ; Assembler loads X: starting at this EEPROM address
RD_X EQU    $C600  ; DSP reads X: from this EEPROM address
APL_ADR EQU    $F0   ; Starting P: address of application program
APL_LEN EQU    $200-APL_ADR ; Maximum length of application program

; Define DSP port addresses
WRLATCH EQU    $FFC1 ; Write to timing board latch
WRSS EQU    $FF80 ; Write clock driver and VP switch states
WRPC EQU    $FFC0 ; Write DSP datum to PCI board
BCR EQU    $FFFE ; Bus (=Port A) Control Register -> Wait States
PBC EQU    $FFE0 ; Port B Control Register
PBDDR EQU    $FFE2 ; Port B Data Direction Register
PBD EQU    $FFE4 ; Port B Data Register
PCC EQU    $FFE1 ; Port C Control Register
PCDDR EQU    $FFE3 ; PortC Data Direction Register
PCD EQU    $FFE5 ; Port C Data Register
IPR EQU    $FFFF ; Interrupt Priority Register
SCR EQU    $FFF0 ; SCI Control Register
SSR EQU    $FFF1 ; SCI Status Register
SCCR EQU    $FFF2 ; SCI Clock Control Register
SRX EQU    $FFF4 ; SCI receive data register
SSITX EQU    $FFEF ; SSI Transmit and Receive data register
CRA EQU    $FFEC ; SSI Control Register A
CRB EQU    $FFED ; SSI Control Register B
TCSR EQU    $FFDE ; Timer control and status register
TCR EQU    $FFDF ; Timer count register
TIM_BIT EQU    0 ; Timer status bit

; Camera operational mode bit definitions
COM_MOD EQU    0 ; Clear if just waiting form commands to interpret
RST_MOD EQU    1 ; Set to continuously reset array
VID1_MOD EQU    2 ; Set if in video mode #1
VID2_MOD EQU    3 ; Set if in video mode #2

```

```

; After RESET jump to initialization code
ORG P:RST_ISR,P:RST_ISR+ROM_OFF
JMP <INIT      ; Initialize DSP after hardware reset
NOP

; The SCI interrupts when it receives data from the PCI board.
ORG P:SCI_ISR,P:SCI_ISR+ROM_OFF
JSR <SCI_RCV   ; Jump to long interrupt service routine
NOP

; The SCI interrupts to here when there is an error.
ORG P:SCI_ERR,P:SCI_ERR+ROM_OFF
JSR <CLR_SCI
NOP

; DSP Timer interrupt for exposure time control
ORG P:TIM_ISR,P:TIM_ISR+ROM_OFF
JSR <TIMER     ; Long interrupt service routine
NOP

; Put the ID words for this version of the ROM code. It is placed at
; the address of the SWI = software interrupt, which we never use.
ORG P:ROM_ID,P:ROM_ID+ROM_OFF
DC $010101     ; Institution: University of Calgary
                ; Location : RAO
                ; Instrument : IR Camera
DC $030002     ; Version 2.30, board #2 = timing

*****
;
;
;
; Permanent address register assignments
; R1 - Address of current contents of PCI board receiver
; R2 - Address of processed contents of PCI board receiver
; R3 - Address of current contents of PCI board transmitter
; R4 - Address of processed contents of PCI board transmitter
; R6 - CCD clock driver address for CCD #0
; It is also the A/D address of analog board #0
; R7 - Return address after exposure calls, may be used sparingly
;
;
; Other registers
; R0, and R5 - Temporary registers used all over the place
*****

```

*

```

; Initialization code is in the application area since it executes only once
    ORG    P:APL_ADR,P:APL_ADR+ROM_OFF    ; Download address

; Define this as simple jump addresses so bootrom program is sure to work
; until the application program can be loaded
APPLICATION
    JMP    <TST_RCV    ; Defined so compiler has APPLICATION address

; Initialization of the DSP - system register, serial link, interrupts.
; This is executed once on DSP boot from ROM, and is not incorporated
; into any download code since its not needed.

INIT  MOVEC    #$0002,OMR    ; Operating Mode Register = Normal
                                ; Expanded - set after reset by hardware

    ORI    #$03,MR    ; Temporarily mask interrupts

    MOVEP    #0,X:PBC    ; Set Port B to general purpose I/O
    MOVEP    #$3FFF,X:PBDDR    ; Set PB0 to PB14 to outputs -
                                ; H0, AUX4, TXD_EN, RXD_EN, STATUS0 to
                                ; STATUS3, AUX1, LVEN, AUX3, FRAME,
                                ; LINE, AUX2. PWRST is an input.

    MOVEP    #$020D,X:PBD    ; RXD-EN = TXD-EN = 1 for enabling PCI
                                ; communication. H0 = 1 to communicate
                                ; with analog boards. LVEN = 1.
                                ; All others = 0.

    MOVEP    #$6002,X:CRA    ; SSI programming - no prescaling;
                                ; 24 bits/word; on-demand communications;
                                ; no prescale; 3.12 MHz serial clock rate

    MOVEP    #$3930,X:CRB    ; SSI programming - OF0, OF1 don't apply;
                                ; SC0, SC1, SC2 are inputs; SCK is output;
                                ; shift MSB first; rcv and xmt asynchronous
                                ; wrt each other; gated clock; bit frame
                                ; sync; network mode to get on-demand;
                                ; RCV and its interrupts enabled; TX enabled,
                                ; TX interrupts disabled -> Utility board SSI

    MOVEP    #$0B02,X:SCR    ; SCI programming: 10-bit asynchronous

```

```

; protocol (1 start, 8 data, no parity,
; 1 stop); LSB before MSB; enable receiver
; and its interrupts; transmitter interrupts
; disabled.

MOVEP #$0050,X:SCCR ; SCI clock: asynchronous data rate =
; 9600 kbits/sec, internal clock.
; (50 MHz / 64 / 81 = 9645 baud)

MOVEP #$0013,X:PCC ; Port C implemented as enabling the SCI
; pins RXD and TXD and HVEN. The SSI will
; be enabled only as needed.

MOVEP #$0013,X:PCD ; Port C Data Register - Set all lines high
; if configured as outputs.

MOVEP #$007F,X:PCDDR ; Port C Data Direction register - Set all
; lines to outputs when not used for SSI
; or SCI service except SRD and STD that
; are pulled low by 500 ohms.

MOVEP #$0181,X:BCR ; Wait states = X: Y: P: and Y: ext. I/O

MOVEP #>2,X:TCSR ; Enable timer interrupts

MOVEP #$61A8,X:TCR ; Divide so timer interrupts every millisecond

; Initialize X: data memory
MOVE #RD_X,R0 ; Starting X: address in EEPROM
MOVE #0,R1 ; Put values starting at beginning of X:
DO #$100,X_MOVE ; Assume 256 = $100 values exist
DO #3,X_LOOP ; Reconstruct bytes to 24-bit words
MOVE P:(R0)+,A2 ; Get one byte from EEPROM
REP #8
ASR A ; Shift right 8 bits
X_LOOP
MOVE A1,X:(R1)+ ; Write 24-bit words to X: memory
X_MOVE

; Initialize registers
MOVE #RCV_BUF,R1 ; Starting address of receiver buffer
MOVE #XMT_BUF,R3 ; Starting address of transmitter buffer

```

```

    MOVE  #WRSS,R6      ; Address of clock and video processor switches
    MOVE  R1,R2
    CLR   A R3,R4
    MOVE  #31,M1        ; All address registers are circular, modulo 32
    MOVE  M1,M2
    MOVE  M2,M3
    MOVE  M3,M4
    MOVE  M1,N1
    DO    #32,ZERO_X    ; Zero all buffers
    MOVE  A,X:(R1)+
    MOVE  A,X:(R3)+
ZERO_X

; Disable analog board functions
    MOVEP    X:LATCH,Y:WRLATCH

; Call Load Application #1 to get video mode loaded on boot as the default
    MOVE  #'LDA',A
    MOVE  A,X:(R1)+
    MOVE  X:<ONE,A
    MOVE  A,X:(R1)+
    MOVE  X:<CAR_RET,A
    MOVE  A,X:(R1)+

; Set interrupt priority levels
    MOVEP  #$038000,X:IPR ; Write to interrupt priority register
                                ; Exposure timer = 2
                                ; SCI = 1 = PCI board link
                                ; Host, IRQA, IRQB all disabled
    ANDI  #$FC,MR        ; Unmask all interrupt levels

; Go execute the program - initialization is over
    JMP   <CHK_HDR      ; Process the commands on the stack

; Check for program space overflow
    IF    @CVS(N,*)>$1FF
    WARN  'Internal P: memory overflow!' ; Don't overflow DSP P: space
    ENDIF

; ***** End of initialization code *****

; Put some of the code in the interrupt vector area that is not used,
; from $18 to $3B (PGM_STR), then continue on at $3E (PGM_CON).

```

```

        ORG    P:PGM_STR,P:PGM_STR+ROM_OFF    ; Program start

; Test serial receiver contents
START    JSET    #TIM_BIT,X:TCSR,CHK_TIM ; If timing down go elsewhere
        JSET    #COM_MOD,X:STATUS,APPLICATION

TST_RCV JSR      <GET_RCV    ; Get a command from the receiver stack
        JEQ     <START      ; If none, test for timer and application

; Process the receiver entry - is it in the command table?
CHK_HDR  MOVE    X:(R2)+,A    ; Get the command buffer entry
        MOVE    #<COM_TBL,R0 ; Get command table starting address
        DO      #NUM_COM,END_COM ; Loop over command table
        MOVE    X:(R0)+,X1    ; Get the command table entry
        CMP     X1,A X:(R0),R5 ; Does receiver = table entry?
        JNE     <NOT_COM      ; No, keep looping
        ENDDO      ; Yes, restore the DO loop system registers

; Wait for the complete command and then jump to it
TST_END  MOVE    X:(R1+N1),A  ; Get most recent SCI word
        MOVE    X:<CAR_RET,X0
        CMP     X0,A          ; Is it = " _CR" ?
        JNE     <TST_END      ; No -> keep waiting

        JMP     (R5)          ; Yes -> execute the command
NOT_COM  MOVE    (R0)+        ; Increment the register past the table address
END_COM

; It's not in the command table - send an error message
ERROR    MOVE    X:<ERR,X0    ; Send an error message 'ERR'
        JMP     <FINISH2

; Construct a simple reply for the PCI board
FINISH    MOVE    (R2)+        ; Step over Carriage Return delimiter
END_EXP   MOVE    X:<DON,X0    ; Send a 'DON' as a reply
FINISH2   MOVE    X0,X:(R3)+  ; Put on the buffer to be transmitted

; Process transmitter buffer to see if anything needs to be sent
PRC_XMT  MOVE    R4,A          ; Address of processed transmitter contents
        MOVE    R3,X0          ; Address of current transmitter contents
        CMP     X0,A          ; Are they equal?
        JEQ     <START        ; If equal, look for receiver contents
        JMP     <XMIT         ; Needed because we're inserting timer ISR

```

```

; Check contents of receiver stack to see if a new host command has come in
GET_RCV  MOVE  R2,X0      ; Get address of processed receiver contents
        MOVE  R1,A        ; Get address of current receiver contents
        CMP   X0,A
        RTS

; Jump here on RRR and M_RA commands so R2 steps over CR delimiter
XMT_DON  MOVE      (R2)+   ; Step over "__CR" delimiter in command
        RTS

; Check for program space overflow
        IF    @CVS(N,*)>$3C
        WARN  'Error: Timer ISR overwritten at P:$3C'
        ENDIF

        ORG   P:PGM_CON,P:PGM_CON+ROM_OFF ; Step over timer ISR

; Transmit the 24-bit word to the PCI board three bytes at a time
XMIT  MOVE  X:<SRXFST,R0 ; R0 = $FFF6 = SCI first byte address
      MOVE  X:(R4)+,A
      DO    #3,SCI_SPT
SCI_XMT JCLR  #0,X:SSR,SCI_XMT ; Continue only if SCI XMT register is empty
      MOVE  A,X:(R0)- ; Write to SCI buffer, increment byte pointer
SCI_SPT

SCI_CR JCLR  #0,X:SSR,SCI_CR ; Continue only if SCI XMT register is empty
      MOVEP #0D,X:SRX ; Transmit a Carriage Return
      JMP   <PRC_XMT

; Start up the exposure timer and wait here until it is done
EXPOSE MOVE  X:<EXP_TIM,A ; Enter exposure time into timer's
      MOVE  A,X:<TGT_TIM ; target time
      CLR   A ; Zero out elapsed time
      MOVE  A,X:<EL_TIM
      BSET  #0,X:TCSR ; Enable DSP timer
CHK_COM JSR   <GET_RCV ; Check for incoming commands
      JNE   <CHK_HDR ; If received, process it normally
CHK_TIM JSET  #0,X:TCSR,CHK_COM ; Wait for timer to end
      JMP   (R7) ; Jump to the internal jump address

; Interrupt service routine for the SCI serial link to the PCI board
SCI_RCV MOVEC SR,X:<SAVE_SR ; Save Status Register
      MOVE  R0,X:<SAVE_R0 ; Save R0

```

```

MOVE B1,X:<SAVE_B1 ; Save B1
MOVE X1,X:<SAVE_X1 ; Save X1
MOVE X:<SCI_R0,R0 ; Get previous value of SCI R0
MOVE X:<SCI_B1,B1 ; Get previous value of SCI B1
MOVE X:(R0),X1 ; Get the byte from the SCI
OR X1,B(R0)- ; Add byte into B1, postdecrement R0
BTST #1,R0 ; Test for the address being $FFF3 = last byte
JCC <MID_BYT ; Not the last byte => only restore registers
END_BYT MOVE B1,X:(R1)+ ; Put the 24-bit word in the command buffer
MOVE X:<SRXFST,R0 ; Initialize R0 most significant byte of SCI
MOVE #0,B1 ; Zero SCI_B1 for next SCI use
MID_BYT MOVE R0,X:<SCI_R0 ; Save SCI value of SCI address pointer
MOVE B1,X:<SCI_B1 ; Save SCI_B1 for next SCI use
MOVEC X:<SAVE_SR,SR ; Restore Status Register
MOVE X:<SAVE_R0,R0 ; Restore R0
MOVE X:<SAVE_B1,B1 ; Restore B1
MOVE X:<SAVE_X1,X1 ; Restore X1
RTI ; Return from interrupt service

```

; Interrupt service routine for the DSP timer, called every millisecond

```

TIMER MOVEC SR,X:<SV_SR ; Save Status Register
MOVE B1,X:<SV_B1
MOVE Y1,X:<SV_Y1
MOVE X:<ONE,B
MOVE X:<EL_TIM,Y1 ; Get elapsed time
ADD Y1,B X:<TGT_TIM,Y1 ; Get target time
MOVE B,X:<EL_TIM ; EL_TIM = EL_TIM + 1
CMP Y1,B
JLT <NO_TIM ; If (EL .GE. TGT) we've timed out
BCLR #0,X:TCSR ; Disable timer
NO_TIM MOVEC X:<SV_SR,SR ; Restore Status Register
MOVE X:<SV_B1,B1
MOVE X:<SV_Y1,Y1
RTI ; Return from TIMER interrupt

```

; Read DSP or EEPROM memory ('RDM' address): read memory, reply with value

```

RDMEM MOVE X:(R2),R0 ; Need the address in an address register
MOVE X:(R2)+,A ; Need address also in a 24-bit register
JCLR #20,A,RDX ; Test address bit for read from P: memory
MOVE P:(R0),X0 ; Read from Program memory
JMP <FINISH2 ; Send out a header ID with the value
RDX JCLR #21,A,RDY ; Test address bit for read from X: memory
MOVE X:(R0),X0 ; Write to X data memory

```

```

        JMP <FINISH2      ; Send out a header ID with the value
RDY  JCLR #22,A,RDR      ; Test address bit for read from Y: memory
      MOVE Y:(R0),X0      ; Read from Y data memory
      JMP <FINISH2      ; Send out a header ID with the value
RDR  JCLR #23,A,ERROR     ; Test address bit for read from EEPROM memory
      MOVE X:<THREE,X0    ; Convert to word address to a byte address
      MOVE R0,Y0          ; Get 16-bit address in a data register
      MPY X0,Y0,A         ; Multiply
      ASR A               ; Eliminate zero fill of fractional multiply
      MOVE A0,R0          ; Need to address memory
      BSET #15,R0         ; Set bit so its in EEPROM space
      DO #3,L1RDR
      MOVE P:(R0)+,A2     ; Read each ROM byte
      REP #8
      ASR A               ; Move right into A1
L1RDR
      MOVE A1,X0          ; FINISH2 transmits X0 as its reply
      JMP <FINISH2

; Program WRMEM ('WRM' address datum): write to memory, reply 'DON'
WRMEM MOVE X:(R2),R0      ; Get the desired address
      MOVE X:(R2)+,A      ; We need a 24-bit version of the address
      MOVE X:(R2)+,X0     ; Get datum into X0 so MOVE works easily
      JCLR #20,A,WRX      ; Test address bit for write to P: memory
      MOVE X0,P:(R0)      ; Write to Program memory
      JMP <FINISH
WRX  JCLR #21,A,WRX       ; Test address bit for write to X: memory
      MOVE X0,X:(R0)      ; Write to X: memory
      JMP <FINISH
WRY  JCLR #22,A,WRR       ; Test address bit for write to Y: memory
      MOVE X0,Y:(R0)      ; Write to Y: memory
      JMP <FINISH
WRR  JCLR #23,A,ERROR     ; Test address bit for write to EEPROM
      MOVE X:<THREE,X1    ; Convert to word address to a byte address
      MOVE R0,Y0          ; Get 16-bit address in a data register
      MPY X1,Y0,A         ; Multiply
      ASR A               ; Eliminate zero fill of fractional multiply
      MOVE A0,R0          ; Need to address memory
      BSET #15,R0         ; Set bit so its in EEPROM space
      MOVE X0,A1          ; Get data from command string
      DO #3,L1WRR        ; Loop over three bytes of the word
      MOVE A1,P:(R0)+     ; Write each EEPROM byte
      REP #8

```

```

        ASR  A X:<C50000,Y0 ; Move right one byte, enter delay
        DO   Y0,L2WRR      ; Delay by 12 milliseconds for EEPROM write
        REP  #4             ; Assume 50 MHz DSP56002
        NOP
L2WRR
        NOP                ; DO loop nesting restriction
L1WRR
        JMP  <FINISH

; Read EEPROM code into DSP memory starting at P:APL_ADR - as a subroutine
LDAPPL MOVE X:(R2)+,X0      ; Number of application program
        MOVE X:<C600,Y0
        MPY  X0,Y0,A X:<ZERO,X1
        ASR  A X:<C300,X0
        SUB  X,A #APL_ADR,R7
        MOVE A0,R0          ; EEPROM address = # x $600 - $300
        BSET #15,R0         ; All EEPROM accesses are with A15=1
        DO   #APL_LEN,LD_LA2 ; Load from APL_ADR to $200
        DO   #3,LD_LA1
        MOVE P:(R0)+,A2     ; Read from EEPROM
        REP  #8
        ASR  A
LD_LA1
        MOVE A1,P:(R7)+     ; Write to DSP P: memory
LD_LA2

; Splice the application and boot command tables together
        MOVE #COM_TBL,R7    ; Leave most of X: memory alone
        DO   #32,LD_LA4      ; 16 commands, 2 entries per command
        DO   #3,LD_LA3
        MOVE P:(R0)+,A2     ; Read from EEPROM
        REP  #8
        ASR  A
LD_LA3
        MOVE A1,X:(R7)+     ; Write to DSP X: memory
LD_LA4

; Transfer Y: memory, containing waveforms and readout parameters
        MOVE #0,R7          ; Start at bottom of Y: memory
        DO   #200-32-APL_LEN,LD_LA6 ; Update Y: DSP memory
        DO   #3,LD_LA5
        MOVE P:(R0)+,A2     ; Read from EEPROM
        REP  #8

```

```

        ASR    A
LD_LA5
        MOVE   A1,Y:(R7)+    ; Write to DSP Y: memory
LD_LA6
        JMP    <FINISH      ; Return and send 'DON'

; Reset = Reboot
RST    RESET          ; Reset peripherals
        MOVE   X:<CFFFF,M0    ; Insure that its linear addressing
        MOVE   X:<CFFFF,M1
        MOVEP  X:ZERO,X:IPR    ; Clear Interrupt Priority Register
        MOVEP  X:CFFFF,X:BCR    ; Many Wait States for PROM accesses
        MOVEC  X:<ZERO,SP      ; Clear the stack pointer
        MOVEC  X:<C300,SR      ; Clear the Condition Code Register
        MOVEC  #S01,OMR      ; Operating Mode Register = Reboot
        NOP                    ; Allow one cycle delay for the remapping
        JMP    <$0            ; Begin bootstrap from internal ROM

; Clear error condition and interrupt on SSI receiver
CLR_SCI MOVEP  X:SSR,X:RCV_ERR ; Read SCI status register
        MOVEP  X:SRX,X:RCV_ERR ; Read register to clear error
        RTI

; Check for program space overflow into application code area
        IF     @CVS(N,*)>=APL_ADR
        WARN   'ERROR - Boot program overflows into application code area'
        ENDIF

; ***** Beginning of X: definitions *****

; Status and header ID processing words
        ORG    X:0,X:LD_X
STATUS DC     0      ; Status word

; Timer related constants
SV_SR      DC     0      ; Save for timer ISR
SV_B1      DC     0      ; Save for timer ISR
SV_Y1      DC     0      ; Save for timer ISR

; Definitions for variables needed for the interrupt service routines
SAVE_SR DC     0
SAVE_X1 DC     0
SAVE_B1 DC     0

```

```

SAVE_R0 DC    0
SCI_B1 DC     0
SCI_R0 DC     $FFF6 ; Current address of the SCI
SRXFST DC     $FFF6 ; Address of first byte in SCI receiver
SPARE    DC    0

; EXP_TIM must be at address $E for compatability with IR Labs' IREM
EL_TIM   DC    0      ; Elapsed exposure time in milliseconds
TGT_TIM  DC    0      ; TGT_TIM = EXP_TIM at beginning of exposure
EXP_TIM  DC    1000   ; Exposure time (milliseconds), written by host computer

; Definition of value in latch U25
LATCH    DC    $10    ; Value in latch chip U25

; Miscellaneous constant definitions
ZERO DC    0
ONE  DC    1
TWO  DC    2
THREE DC    3
EN_SI DC    $0173      ; Enable the SCI and SSI pins
DISA_SI DC    $0003
C300 DC    $300        ; Constant for resetting the DSP
C600 DC    $600        ; EEPROM space per application program
CFFFF DC    $FFFF      ; Constant for resetting the DSP
C50000 DC    50000      ; Delay for WRROM = 12 millisec
ERR  DC    'ERR'        ; An error occurred
DON  DC    'DON'        ; Command was fully processed
RCV_ERR DC    0          ; Dummy location for receiver clearing
CAR_RET DC    $20200D    ; Carriage Return marking end of command

; Command table resident in X: data memory
; The last part of the command table is not defined for "bootrom"
; because it contains application-specific commands

ORG    X:COM_TBL,X:COM_TBL+LD_X
DC    0,START,0,START,0,START,0,START ; This is where application
DC    0,START,0,START,0,START,0,START ; commands go
DC    0,START,0,START,0,START,0,START
DC    0,START,0,START,0,START,0,START
DC    'ERR',START ; Nothing special
DC    'RDM',RDMEM ; Read from DSP or EEPROM memory
DC    'WRM',WRMEM ; Write to DSP memory
DC    'LDA',LDAPPL ; Load application program from EEPROM to DSP

```

```
DC  'RST',RST      ; Re-boot DSP from on-board ROM
DC  'STP',FINISH   ; Put it here as a no op
DC  $20200D,START  ; Extra delimiters - do nothing
DC  0,START        ; Room for one more command
```

```
; End of command table
```

```
; End of program
END
```

D.2.2 App.asm as supplied by IR Labs

COMMENT *

This file is used to generate DSP code for the second generation timing boards to operate a PICNIC infrared array.
 Fiber optic and PCI application files are now joined into one.
 File modified 11/97 to generate timing waveforms similar to old versions of PICNIC delivered with the microscopes
 Changed Aug. '98 to control Rev. 6C power control board, and use extended on-board SRAM

Base code University of Calgary.
 Code is proved to work. Date is 010705
 Required command-line switches, DL = 0; FOPCI = 0;

-d DOWNLOAD 1 To generate code for downloading to DSP memory.
 -d DOWNLOAD 0 To generate code for writing to the EEPROM.

*

PAGE 132 ; Printronix page width - 132 columns

; Define a section name so it doesn't conflict with other application programs
 SECTION TIMIR

; These are also defined in "timboot.asm", so be sure they agree
 APL_NUM EQU 1 ; Application number from 1 to 10
 APL_ADR EQU \$F0 ; P: memory location where application code begins
 APL_LEN EQU \$200-APL_ADR ; Maximum length of application program
 COM_TBL EQU \$A0 ; Starting address of command table in X: memory
 TBL_ADR EQU \$0F ; Waveform tables starting address

; Define some timing board addresses and bit numbers
 WRFO EQU \$FFC0 ; Write to fiber optic serial transmitter
 WRLATCH EQU \$FFC1 ; Write to timing board latch
 SSITX EQU \$FFEF ; SSI Transmit and Receive data register
 PCC EQU \$FFE1 ; Port C Control Register
 PBD EQU \$FFE4 ; Port B Data Register

```

TCSR EQU $FFDE ; Timer control and status register
CDAC EQU 0 ; Bit number in U25 for clearing DACs
WW EQU 1 ; Word width of serial data
ENCK EQU 2 ; Bit number in U25 for enabling analog switches
;LVEN EQU 2 ; Low voltage enable (+/- 6.5, +/- 15 volt nominal)
;HVEN EQU 3 ; High voltage enable (+36 volts, only used for reset)

; Values for timPC board
LVEN EQU 9 ; Low voltage enable (+/- 6.5, +/- 15 volt nominal)
HVEN EQU 4 ; High voltage enable (+36 volts, only used for reset)

; Specify execution and load addresses
IF DL
ORG P:APL_ADR,P:APL_ADR ; Download address
ELSE
ORG P:APL_ADR,P:(2*APL_NUM-1)*$100 ; EEPROM generation
ENDIF

APPLICATION
JSET #RST_MOD,X:STATUS,CONT_RST
JSET #VID1_MOD,X:STATUS,VIDEO_MODE1
JSET #VID2_MOD,X:STATUS,VIDEO_MODE2
JMP <TST_RCV

; Set software to video mode #1
VD1 BSET #VID1_MOD,X:<STATUS
BCLR #VID2_MOD,X:<STATUS
BCLR #RST_MOD,X:STATUS ; Continuous reset mode off
JMP <FINISH ; Send reply

; Set software to video mode #2
VD2 BSET #VID2_MOD,X:<STATUS
BCLR #VID1_MOD,X:<STATUS
BCLR #RST_MOD,X:STATUS ; Continuous reset mode off
JMP <FINISH ; Send reply

; Exit video mode, enter continuous reset mode
STP BSET #RST_MOD,X:STATUS ; Continuous reset mode on
BCLR #VID1_MOD,X:<STATUS
BCLR #VID2_MOD,X:<STATUS
JMP <FINISH

; Video mode #1 - reset, integrate, read, ad infinitum

```

```

VIDEO_MODE1
    MOVE #NO_CHK,R5 ; Don't process incoming commands
    JSR <RESET_ARRAY ; Reset the array twice
    MOVE #L_VID1,R7 ; Return address after exposure
    JMP <EXPOSE ; Delay for specified exposure time
L_VID1
    JCLR #VID1_MOD,X:STATUS,TST_RCV ; Exit video mode
    JSR <RD_ARRAY ; Read the array
    JMP <TST_RCV ; Look for a new command

; Video mode #2 - reset, short delay, read, integrate, read, ad infinitum
VIDEO_MODE2
    MOVE #NO_CHK,R5 ; Don't process incoming commands
    JSR <RESET_ARRAY ; Reset the array
    JSR <SHORT_DELAY ; Call short delay for reset to settle down
    JSR <RD_ARRAY ; Read the array
    MOVE #L_VID2,R7 ; Return address after exposure
    JMP <EXPOSE ; Delay for specified exposure time
L_VID2
    JCLR #VID2_MOD,X:STATUS,TST_RCV ; Exit video mode
    JSR <RD_ARRAY ; Read the array
    JMP <TST_RCV ; Look for a new command

; Continuously reset array, checking for host commands every line
CONT_RST
    MOVE #<GET_RCV,R5
    JSR <RESET_ARRAY
    JNE <CHK_HDR
    JMP <CONT_RST

; Set the exposure time
SET_EXT MOVE X:(R2)+,A ; Get third word of command = exposure time
    MOVE #>5,X0 ; Subtract 5 millisec from exposure time to
    SUB X0,A ; account for READ to FSYNC delay time
    MOVE A,X:<EXP_TIM ; Write to magic address
    JMP <FINISH ; Send out 'DON' reply

; Short delay for the array to settle down after a global reset
SHORT_DELAY
    MOVE Y:<RST_DLY,A ; Enter reset delay into timer
CON_DELAY ; Alternate entry for camera on delay
    MOVE A,X:<TGT_TIM
    CLR A ; Zero out elapsed time

```

```

    MOVE A,X:<EL_TIM
    BSET #0,X:TCSR      ; Enable DSP timer
    CNT_DWN JSET #0,X:TCSR,CNT_DWN ; Wait here for timer to count down
    RTS

; Abort exposure and stop the timer
ABR_EXP CLR A      ; Just stop the timer
    MOVE A,X:<TGT_TIM
    JMP <FINISH    ; Send normal reply

; Dummy subroutine to not call receiver checking routine
NO_CHK BCLR #0,SR ; Clear status register clear bit
    RTS

; Reset entire array and don't transmit any pixel data
RESET_ARRAY
    MOVE #<READ_ON,R0 ; Turn Read ON
    JSR <CLOCK
    DO Y:<N_RSTS,L_RESET
    MOVE #<FRAME_INIT,R0
    JSR <CLOCK
    DO #64,END_FRAME
    MOVE #<SHIFT_RESET_ODD_ROW,R0 ; Shift and reset the line
    JSR <CLOCK
    DO #64,L_ODD
    MOVE #<SHIFT_ODD_ROW_PIXELS,R0
    JSR <CLOCK
    NOP
L_ODD
    MOVE #<SHIFT_RESET_EVEN_ROW,R0 ; Shift and reset the line
    JSR <CLOCK
    DO #64,L_EVEN
    MOVE #<SHIFT_EVEN_ROW_PIXELS,R0
    JSR <CLOCK
    NOP
L_EVEN
    JSR (R5) ; Check for incoming command if in continuous
    JEQ <NOT_COM ; reset mode
    ENDDO ; If there is an incoming command then exit
    ENDDO ; continuous mode and return
    JMP <END_RST
NOT_COM NOP
END_FRAME NOP

```

```

L_RESET NOP      ; End of loop label for reading rows
END_RST MOVE #<READ_OFF,R0 ; Turn Read OFF
JSR <CLOCK
RTS      ; Return from subroutine call

; ***** ARRAY READOUT *****
RD_ARRAY
IF FOPCI      ; Optionally send "RDA" to FO-PCI board
MOVEP Y: CBD_HDR,Y:WRFO
JSR <PAL_DLY
MOVEP Y:RDA,Y:WRFO
JSR <PAL_DLY
MOVEP Y:NPXLS,Y:WRFO
ENDIF

BSET #WW,X:PBD ; Set WW to 1 for 16-bit image data
MOVE #<READ_ON,R0 ; Turn Read ON and wait 5 milliseconds
JSR <CLOCK      ; so first few rows aren't at high
DO #598,DLY_ON ; count levels
JSR <PAL_DLY
NOP
DLY_ON

MOVE #<FRAME_INIT,R0 ; Initialize the frame for readout
JSR <CLOCK

DO #64,FRAME

; First shift and read the odd numbered rows
MOVE #<SHIFT_ODD_ROW,R0 ; Shift odd numbered rows
JSR <CLOCK
MOVE #<SHIFT_ODD_ROW_PIXELS,R0 ; Shift 2 columns, no transmit
JSR <CLOCK

DO #63,L_ODD_ROW
MOVE #<READ_ODD_ROW_PIXELS,R0 ; Read the pixels in odd rows
JSR <CLOCK
NOP
L_ODD_ROW
MOVE #<SXMIT_EIGHT_PIXELS,R0 ; Series transmit last 8 pixels
JSR <CLOCK

; Then shift and read the even numbered rows

```

```

MOVE #<SHIFT_EVEN_ROW,R0 ; Shift even numbered rows
JSR <CLOCK
MOVE #<SHIFT_EVEN_ROW_PIXELS,R0 ; Shift 2 columns, no transmit
JSR <CLOCK

DO #63,L_EVEN_ROW
MOVE #<READ_EVEN_ROW_PIXELS,R0 ; Read the pixels in even rows
JSR <CLOCK
NOP
L_EVEN_ROW
MOVE #<SXMIT_EIGHT_PIXELS,R0 ; Series transmit last 8 pixels
JSR <CLOCK
NOP
FRAME
MOVE #<READ_OFF,R0 ; Turn Read Off
JSR <CLOCK
JSR <PAL_DLY ; Wait for serial data transmission
BCLR #WW,X:PBD ; Clear WW to 0 for non-image data
RTS

; ***** Acquire a complete image *****
; Call multiple read array with number of read pairs = 1
RRR MOVE X:<ONE,A
MOVE A,Y:<N_RA
JMP <L_MRA0

; Reset array, wait, read it out n times, expose, read it out n times
M_RA MOVE X:(R2)+,A
MOVE A,Y:<N_RA ; Desired number of reset/read pairs
L_MRA0 JSR <XMT_DON ; Temporarily transmit 'DON' for FO case only
MOVE #NO_CHK,R5 ; Don't check for incoming commands
JSR <RESET_ARRAY ; Reset the array twice
JSR <SHORT_DELAY ; Call short delay for reset to settle down
DO Y:<N_RA,L_MRA1 ; Read N_RA times
JSR <RD_ARRAY ; Call read array subroutine
NOP
L_MRA1
MOVE #L_MRA2,R7
JMP <EXPOSE ; Delay for specified exposure time
L_MRA2 DO Y:<N_RA,L_MRA3 ; Read N_RA times again
JSR <RD_ARRAY ; Call read array subroutine
NOP
L_MRA3

```

```

JMP <END_EXP ; This is the end of the exposure

; ***** SUBROUTINES *****
; Core subroutine for clocking out array charge
CLOCK MOVE Y:(R0)+,X0 ; # of waveform entries
      MOVE Y:(R0)+,A ; Start the pipeline
      DO X0,CLK1 ; Repeat X0 times
      MOVE A,X:(R6) Y:(R0)+,A ; Send out the waveform
CLK1
      MOVE A,X:(R6) ; Flush out the pipeline
      RTS ; Return from subroutine

; Update the DACs
SET_DAC DO Y:(R0)+,SET_L0 ; Repeat X0 times
      MOVEP Y:(R0)+,X:SSITX ; Send out the waveform
      JSR <PAL_DLY ; Wait for SSI and PAL to be empty
      NOP ; Do loop restriction
SET_L0
      RTS ; Return from subroutine

; Delay for serial writes to the PALs and DACs by 8 microsec
PAL_DLY DO #200,DLY ; Wait 8 usec for serial data transmission
      NOP
DLY NOP
      RTS

; Delay between power control board instructions
DLY_PWR DO #4000,L_PDLY
      NOP
L_PDLY
      RTS

; Set video offsets and DC bias supply voltages
SET_BIAS_NUM
      MOVEP X:EN_SL,X:PCC ; Enable the SSI
      JSR <PAL_DLY ; Wait for the SSI port to be enabled
      MOVE X:(R2)+,A ; First argument is board number, 0 to 15
      REP #20
      LSL A
      MOVE A,X0
      MOVE X:(R2)+,A ; Second argument is DAC number, 0 to 7
      REP #14
      LSL A

```

```

OR X0,A
BSET #19,A1 ; Set bits meaning DAC
BSET #18,A1
MOVE A,X0
MOVE X:(R2)+,A ; Third argument is voltage value
MOVE #$000FFF,Y0 ; Mask off just 12 bits to be sure
AND Y0,A
OR X0,A
MOVEP A,X:SSITX ; Write the number to the DAC
MOVEP X:DISA_SI,X:PCC ; Disable the SSI
JMP <FINISH

; Power off
PWR_OFF MOVEP X:EN_SI,X:PCC ; Enable the SSI
BCLR #CDAC,X:<LATCH ; Clear all DACs
BCLR #ENCK,X:<LATCH ; Disable DAC output switches
MOVEP X:LATCH,Y:WRLATCH
BSET #LVEN,X:PBD ; LVEN = HVEN = 1 => Power reset
; BSET #HVEN,X:PBD ; !!!
BSET #HVEN,X:PCD ; timPC value
BCLR #COM_MOD,X:STATUS ; Command execution mode
BCLR #RST_MOD,X:STATUS ; Continuous reset mode off
MOVEP X:DISA_SI,X:PCC ; Disable the SSI
JMP <FINISH

; Start power-on cycle
CAM_ON MOVEP X:EN_SI,X:PCC ; Enable the SSI
BSET #CDAC,X:<LATCH ; Disable clearing of all DACs
BCLR #ENCK,X:<LATCH ; Disable DAC output switches
MOVEP X:LATCH,Y:WRLATCH

; Turn analog power on to controller boards, but not yet to IR array
BSET #LVEN,X:PBD ; LVEN = HVEN = 1 => Power reset
; BSET #HVEN,X:PBD ; !!!
BSET #HVEN,X:PCD ; timPC value

; Now ramp up the low voltages (+/- 6.5V, 16.5V) and delay them to turn on
BCLR #LVEN,X:PBD ; LVEN = Low => Turn on +/- 6.5V, +/- 16.5V
MOVE Y:<PWR_DLY,A
JSR <CON_DELAY

; Zero all bias voltages and enable DAC output switches
MOVE #<ZERO_BIASES,R0 ; Get starting address of DAC values

```

```

JSR <SET_DAC
MOVE X:<THREE,A
JSR <CON_DELAY
BSET #ENCK,X:<LATCH ; Enable clock and DAC output switches
MOVEP X:LATCH,Y:WRLATCH ; Disable DAC clearing, enable clocks

; Turn on Vdd = digital power unit cell to the IR array
MOVEP Y:VDD,X:SSITX ; pin #5 = Vdd = digital power on array

; Delay for the IR array to settle down
MOVE Y:<VDD_DLY,A ; Delay for the IR array to settle
JSR <CON_DELAY

; Set DC bias DACs
SETBIAS MOVEP X:EN_SI,X:PCC ; Enable the SSI
JSR <PAL_DLY ; Wait for port to be enabled
MOVE #<DC_BIASES,R0 ; Get starting address of DAC values
JSR <SET_DAC
MOVE X:<THREE,A ; Delay three millisec to settle
JSR <CON_DELAY

; Set clock driver DACs
MOVE #<DACs,R0 ; Get starting address of DAC values
JSR <SET_DAC

; Turn continuous reset mode on, disable the SSI, and return
BSET #RST_MOD,X:STATUS ; Put controller in continuous reset mode
BSET #COM_MOD,X:STATUS ; Put controller in camera run mode
MOVEP X:DISA_SI,X:PCC ; Disable the SSI
JMP <FINISH

; Command table
IF DL ; Memory offsets for downloading code
ORG X:COM_TBL,X:COM_TBL
ELSE ; Memory offsets for generating EEPROMs
ORG P:COM_TBL,P:(2*APL_NUM-1)*$100+APL_LEN
ENDIF
DC 'RRR',RRR ; Reset, Read, Read array
DC 'MRA',M_RA ; Multiple reads of array
DC 'ABR',ABR_EXP ; End current exposure
DC 'CON',CAM_ON ; Turn on all camera biases and clocks
DC 'PON',CAM_ON ; Turn on all camera biases and clocks
DC 'POF',PWR_OFF ; Turn +/- 15V power supplies off

```

```

DC 'SET',SET_EXT ; Set exposure time
DC 'SBN',SET_BIAS_NUM ; Set bias number
DC 'SBV',SETBIAS ; Set DC bias supply voltages
DC 'VD1',VD1 ; Put array in video #1 mode
DC 'VD2',VD2 ; Put array in video #2 mode
DC 'STP',STP ; Exit video mode
DC 'DON',START ; Nothing special
DC 0,START,0,START,0,START

IF DL
ORG Y:0,Y:0 ; Download address
ELSE
ORG Y:0,P: ; EEPROM address continues from P: above
ENDIF

DUMMY DC 0 ; Left over from previous versions
NCOLS DC 63 ; Number of columns
NROWS DC 64 ; Number of rows
N_RA DC 1 ; Desired number of reset/read pairs
RST_DLY DC 50 ; Delay after array reset for settling
PWR_DLY DC 100 ; Delay in millise for power to turn on
VDD_DLY DC 300 ; Delay in millise for VDD to settle
N_RSTS DC 2 ; Number of resets
NPXLS DC 65536 ; Number of pixels transmitted per image
CBD_HDR DC $AA0002 ; Header to transmit to converter board
RDA DC 'RDA' ; Read array command

; Start the voltage and timing tables at a fixed address
IF DL
ORG Y:TBL_ADR,Y:TBL_ADR ; Download address
ELSE
ORG Y:TBL_ADR,P:(2*APL_NUM-1)*$100+APL_LEN+47 ; EEPROM address
ENDIF

; Miscellaneous definitions
VIDEO EQU $000000 ; Video board select = 0 for first A/D board with biases
BD2 EQU $002000 ; Clock board select = 2
DELAY EQU $480000 ; Delay for clocking operations
; (20 ns unit, 160 if MSB set)
VP_DLY EQU $2C0000 ; Video delay time for 3 microsec/pixel
SXMIT EQU $00F060 ; Series transmit A/D channels #0 - 3

; Clock voltage definitions

```

```
;CLK_HIGH EQU $CC0 ; ~+4V, assuming +VREF = +2.5
CLK_HIGH EQU $880
CLK_LOW EQU $0F4 ; ~+.30V, assuming -VREF = 0.0
```

```
; Table of offset values begins at Y:$10
```

```
; DAC settings for the video offsets
```

```
DC_BIASES DC ZERO_BIASES-DC_BIASES-1
```

```
OFF_0 DC $0c0000 ; Input offset board #0, channel A
```

```
OFF_1 DC $0c4000 ; Input offset board #0, channel B
```

```
OFF_2 DC $1c0000 ; Input offset board #1, channel A
```

```
OFF_3 DC $1c4000 ; Input offset board #1, channel B
```

```
; DAC settings to generate DC bias voltages for the PICNIC array,
```

```
; assuming +7.5 volts maximum from each bias circuit
```

```
VOFFSET DC $0c87e4 ; pin #1 = preamp offset = +3.7 volts
```

```
VRESET DC $0cc111 ; pin #2 = reset = +0.5 volts
```

```
VD DC $0d0a97 ; pin #3 = analog power = +5.0 volts
```

```
ICTL DC $0d47e0 ; pin #4 = current control = +3.7 volts
```

```
VDD DC $0d8878 ; pin #5 = digital power = +4.0 volts
```

```
VUNUSED DC $0dc000 ; pin #6 = unused to 0V
```

```
; Zero out the DC biases during the power-on sequence
```

```
ZERO_BIASES
```

```
DC DACS-ZERO_BIASES-1
```

```
DC $0c8000 ; Pin #1, board #0
```

```
DC $0cc000 ; Pin #2
```

```
DC $0d0000 ; Pin #3
```

```
DC $0d4000 ; Pin #4
```

```
DC $0d8000 ; Pin #5
```

```
DC $0dc000 ; Pin #6
```

```
DC $1c8000 ; Pin #1, board #1
```

```
DC $1cc000 ; Pin #2
```

```
DC $1d0000 ; Pin #3
```

```
DC $1d4000 ; Pin #4
```

```
DC $1d8000 ; Pin #5
```

```
DC $1dc000 ; Pin #6
```

```
; Initialize all DACs, starting with the clock driver ones
```

```
DACS DC READ_ON-DACS-1
```

```
DC (BD2<<8)+(0<<14)+CLK_HIGH ; Pin #1, RESET
```

```
DC (BD2<<8)+(1<<14)+CLK_LOW
```

```

DC (BD2<<8)+(2<<14)+CLK_HIGH ; Pin #2, LINE
DC (BD2<<8)+(3<<14)+CLK_LOW
DC (BD2<<8)+(4<<14)+CLK_HIGH ; Pin #3, LSYNC
DC (BD2<<8)+(5<<14)+CLK_LOW
DC (BD2<<8)+(6<<14)+CLK_HIGH ; Pin #4, FSYNC
DC (BD2<<8)+(7<<14)+CLK_LOW
DC (BD2<<8)+(8<<14)+CLK_HIGH ; Pin #5, PIXEL
DC (BD2<<8)+(9<<14)+CLK_LOW
DC (BD2<<8)+(10<<14)+CLK_HIGH ; Pin #6, READ
DC (BD2<<8)+(11<<14)+CLK_LOW
DC (BD2<<8)+(12<<14) ; Pin #7, not connected=0 volts
DC (BD2<<8)+(13<<14)
DC (BD2<<8)+(14<<14) ; Pin #8, not connected=0 volts
DC (BD2<<8)+(15<<14)

```

```

; Define switch state bits for the clocks

```

```

L_RST EQU 0
H_RST EQU 1
L_LINE EQU 0
H_LINE EQU 2
L_LSYNC EQU 0
H_LSYNC EQU 4
L_FSYNC EQU 0
H_FSYNC EQU 8
L_PIXEL EQU 0
H_PIXEL EQU $10
L_READ EQU 0
H_READ EQU $20

```

```

; Turn READ ON for readout and reset

```

```

READ_ON
DC READ_OFF-READ_ON-2
DC BD2+H_READ+L_PIXEL+H_LSYNC+L_LINE+H_FSYNC+L_RST
DC BD2+H_READ+L_PIXEL+H_LSYNC+L_LINE+H_FSYNC+L_RST

```

```

; Turn READ OFF during exposure

```

```

READ_OFF
DC SHIFT_RESET_ODD_ROW-READ_OFF-2
DC BD2+L_READ+L_PIXEL+H_LSYNC+L_LINE+H_FSYNC+L_RST+DELAY
DC BD2+L_READ+L_PIXEL+H_LSYNC+L_LINE+H_FSYNC+L_RST+DELAY

```

```

; Shift and reset the odd numbered lines

```

```

SHIFT_RESET_ODD_ROW

```

```

DC SHIFT_RESET_EVEN_ROW-SHIFT_RESET_ODD_ROW-2
DC BD2+H_READ+L_PIXEL+L_LSYNC+L_LINE+H_FSYNC+L_RST+DELAY
DC BD2+H_READ+L_PIXEL+H_LSYNC+L_LINE+H_FSYNC+L_RST+DELAY
DC BD2+H_READ+L_PIXEL+H_LSYNC+H_LINE+H_FSYNC+L_RST+DELAY
DC BD2+H_READ+L_PIXEL+H_LSYNC+H_LINE+H_FSYNC+H_RST+DELAY
DC BD2+H_READ+L_PIXEL+H_LSYNC+H_LINE+H_FSYNC+L_RST+DELAY

```

; Shift and reset the even numbered lines

SHIFT_RESET_EVEN_ROW

```

DC FRAME_INIT-SHIFT_RESET_EVEN_ROW-2
DC BD2+H_READ+L_PIXEL+L_LSYNC+H_LINE+H_FSYNC+L_RST+DELAY
DC BD2+H_READ+L_PIXEL+H_LSYNC+H_LINE+H_FSYNC+L_RST+DELAY
DC BD2+H_READ+L_PIXEL+H_LSYNC+L_LINE+H_FSYNC+L_RST+DELAY
DC BD2+H_READ+L_PIXEL+H_LSYNC+L_LINE+H_FSYNC+H_RST+DELAY
DC BD2+H_READ+L_PIXEL+H_LSYNC+L_LINE+H_FSYNC+L_RST+DELAY

```

; Initialize the frame for readout, including shift register (slow row scanner)

FRAME_INIT

```

DC SHIFT_ODD_ROW-FRAME_INIT-2
DC BD2+H_READ+L_PIXEL+H_LSYNC+L_LINE+L_FSYNC+L_RST+DELAY
DC BD2+H_READ+L_PIXEL+L_LSYNC+L_LINE+L_FSYNC+L_RST+DELAY
DC BD2+H_READ+L_PIXEL+L_LSYNC+L_LINE+H_FSYNC+L_RST

```

SHIFT_ODD_ROW

```

DC READ_ODD_ROW_PIXELS-SHIFT_ODD_ROW-2
DC BD2+H_READ+L_PIXEL+L_LSYNC+L_LINE+H_FSYNC+L_RST+DELAY
DC BD2+H_READ+L_PIXEL+H_LSYNC+L_LINE+H_FSYNC+L_RST+DELAY
DC BD2+H_READ+L_PIXEL+H_LSYNC+L_LINE+H_FSYNC+L_RST+DELAY
DC BD2+H_READ+L_PIXEL+H_LSYNC+H_LINE+H_FSYNC+L_RST+$900000
DC BD2+H_READ+H_PIXEL+H_LSYNC+H_LINE+H_FSYNC+L_RST+$7C0000

```

READ_ODD_ROW_PIXELS

```

DC SHIFT_ODD_ROW_PIXELS-READ_ODD_ROW_PIXELS-2
DC BD2+H_READ+H_PIXEL+H_LSYNC+H_LINE+H_FSYNC+L_RST+DELAY
DC VP_DLY ; A/D sample
DC $010033 ; Start A/D conversion
DC SXMIT ; Series transmit four pixels' data
DC $140033 ; Delay
DC BD2+H_READ+L_PIXEL+H_LSYNC+H_LINE+H_FSYNC+L_RST+DELAY
DC VP_DLY ; A/D sample
DC $010033 ; Start A/D conversion
DC SXMIT ; Series transmit four pixels' data

```

SHIFT_ODD_ROW_PIXELS

```

DC SHIFT_EVEN_ROW-SHIFT_ODD_ROW_PIXELS-2
DC BD2+H_READ+H_PIXEL+H_LSYNC+H_LINE+H_FSYNC+L_RST+DELAY
DC VP_DLY ; A/D sample
DC $010033 ; Start A/D conversion
DC $160033 ; Padding
DC BD2+H_READ+L_PIXEL+H_LSYNC+H_LINE+H_FSYNC+L_RST+DELAY
DC VP_DLY ; A/D sample
DC $010033 ; Start A/D conversion
DC $000033 ; Padding

```

SHIFT_EVEN_ROW

```

DC READ_EVEN_ROW_PIXELS-SHIFT_EVEN_ROW-2
DC BD2+H_READ+L_PIXEL+L_LSYNC+H_LINE+H_FSYNC+L_RST+DELAY
DC BD2+H_READ+L_PIXEL+H_LSYNC+H_LINE+H_FSYNC+L_RST+DELAY
DC BD2+H_READ+L_PIXEL+H_LSYNC+H_LINE+H_FSYNC+L_RST+DELAY
DC BD2+H_READ+L_PIXEL+H_LSYNC+L_LINE+H_FSYNC+L_RST+$900000
DC BD2+H_READ+H_PIXEL+H_LSYNC+L_LINE+H_FSYNC+L_RST+$7C0000

```

READ_EVEN_ROW_PIXELS

```

DC SHIFT_EVEN_ROW_PIXELS-READ_EVEN_ROW_PIXELS-2
DC BD2+H_READ+H_PIXEL+H_LSYNC+L_LINE+H_FSYNC+L_RST+DELAY
DC VP_DLY ; A/D sample
DC $010033 ; Start A/D conversion
DC SXMIT ; Series transmit four pixels' data
DC $140033 ; Delay
DC BD2+H_READ+L_PIXEL+H_LSYNC+L_LINE+H_FSYNC+L_RST+DELAY
DC VP_DLY ; A/D sample
DC $010033 ; Start A/D conversion
DC SXMIT ; Series transmit four pixels' data

```

SHIFT_EVEN_ROW_PIXELS

```

DC SXMIT_EIGHT_PIXELS-SHIFT_EVEN_ROW_PIXELS-2
DC BD2+H_READ+H_PIXEL+H_LSYNC+L_LINE+H_FSYNC+L_RST+DELAY
DC VP_DLY ; A/D sample
DC $010033 ; Start A/D conversion
DC $160033 ; Padding
DC BD2+H_READ+L_PIXEL+H_LSYNC+L_LINE+H_FSYNC+L_RST+DELAY
DC VP_DLY ; A/D sample
DC $010033 ; Start A/D conversion
DC $000033 ; Padding

```

SXMIT_EIGHT_PIXELS

```

DC END_TBL-SXMIT_EIGHT_PIXELS-2
DC DELAY+$000033
DC VP_DLY    ; A/D sample
DC $000033   ; Start A/D conversion
DC SXMIT     ; Series transmit four pixels' data
DC DELAY+$000033
DC VP_DLY    ; A/D sample
DC $000033   ; Start A/D conversion
DC SXMIT     ; Series transmit four pixels' data

END_TBL DC 0    ; End of waveform tables

; Check for overflow in the EEPROM case
IF !DL
    IF @CVS(N,@LCV(L))>((2*APL_NUM+1)*$100)*3
    WARN 'EEPROM overflow!' ; Make sure next application
    ENDIF    ; will not be overwritten
ENDIF

ENDSEC    ; End of section TIMIR

; End of program
END

```

D.2.3 Bootstrap.asm as modified by University of Calgary

COMMENT *

This file is used to generate boot DSP code for the second generation TIMII timing board with the PC interface for IR Labs.
 This is Rev. 3.00 software.
 Overlays are no longer used, but application programs can be loaded.
 Modified starting for downloading operation with timllappl.asm
 Header ID code eliminated since the utility board will not be used -
 may be re-implemented if needed. (Aug. 23, 1996)
 Buffers for commands and replies was simplified to just two buffer, one
 for the receiver, one for the transmitter. Each has an address register
 pointing to the current value of the last entry in the buffer (R1 for
 receiver, R3 for transmitter) and an address register pointing to the
 last processed entry (R2 for the receiver, R4 for the transmitter).
 (Aug. 25, 1996)
 SCI interrupt service routine to place the first character in the incoming
 stream into the most significant byte of the 3-byte DSP word.
 (Aug. 26, 1996)
 Timer code based on DSP timer interrupt service added Aug. 31, 1996. It was
 verified to work by testing the X:TCSR bit 0 = TE for timer complete.
 Modified for Rev. 3 PCI timing boards March '97
 Modified for Rev. 6C power board Aug. '98

Base copy 010705 University of Calgary. Code is proved to work.

Designated institution code 010101

This is code version 0.23

CHANGE HISTORY

2001-07-09

Comment added in serial port receive routine.

2001-07-16

Set APL_LEN to \$300 to allow for max length

■

PAGE 132 ; Printronix page width - 132 columns

; Define some useful DSP register locations

```

RST_ISR EQU $00 ; Hardware reset interrupt
ROM_ID EQU $06 ; Location of program Identification = SWI interrupt
SCI_ISR EQU $14 ; SCI serial receiver interrupt address
SCI_ERR EQU $16 ; SCI interrupt with exception (error)
PGM_STR EQU $18 ; Starting address of program
TIM_ISR EQU $3C ; DSP timer interrupt service routine address
PGM_CON EQU $3E ; Program continues on here
BUF_STR EQU $60 ; Starting address of buffers in X:
BUF_LEN EQU $20 ; Length of each buffer
RCV_BUF EQU BUF_STR ; Starting address of serial receiver buffer in X:
XMT_BUF EQU BUF_STR+BUF_LEN ; Starting address of command buffer in X:
COM_TBL EQU XMT_BUF+BUF_LEN ; Starting address of command table in X:
; NUM_COM EQU 24 ; Number of entries in command table
NUM_COM EQU 32 ; Number of entries in command table

ROM_OFF EQU $4000 ; Boot program offset address in EEPROM
LD_X EQU $4200 ; Assembler loads X: starting at this EEPROM address
RD_X EQU $C600 ; DSP reads X: from this EEPROM address
APL_ADR EQU $F0 ; Starting P: address of application program
APL_LEN EQU $300-APL_ADR ; Maximum length of application program

; Define DSP port addresses
WRLATCH EQU $FFC1 ; Write to timing board latch
WRSS EQU $FF80 ; Write clock driver and VP switch states
WRPC EQU $FFC0 ; Write DSP datum to PCI board
BCR EQU $FFFE ; Bus (=Port A) Control Register -> Wait States
PBC EQU $FFE0 ; Port B Control Register
PBDDR EQU $FFE2 ; Port B Data Direction Register
PBD EQU $FFE4 ; Port B Data Register
PCC EQU $FFE1 ; Port C Control Register
PCDDR EQU $FFE3 ; PortC Data Direction Register
PCD EQU $FFE5 ; Port C Data Register
IPR EQU $FFFF ; Interrupt Priority Register
SCR EQU $FFF0 ; SCI Control Register
SSR EQU $FFF1 ; SCI Status Register
SCCR EQU $FFF2 ; SCI Clock Control Register
SRX EQU $FFF4 ; SCI receive data register
SSITX EQU $FFEF ; SSI Transmit and Receive data register
CRA EQU $FFEC ; SSI Control Register A
CRB EQU $FFED ; SSI Control Register B
TCSR EQU $FFDE ; Timer control and status register
TCR EQU $FFDF ; Timer count register
TIM_BIT EQU 0 ; Timer status bit

```

```

; Camera operational mode bit definitions
COM_MOD EQU 0      ; Clear if just waiting form commands to interpret
RST_MOD EQU 1      ; Set to continuously reset array
VID1_MOD EQU 2     ; Set if in video mode #1
VID2_MOD EQU 3     ; Set if in video mode #2

```

```

; After RESET jump to initialization code
ORG P:RST_ISR,P:RST_ISR+ROM_OFF
JMP <INIT      ; Initialize DSP after hardware reset
NOP

```

```

; The SCI interrupts when it receives data from the PCI board.
ORG P:SCI_ISR,P:SCI_ISR+ROM_OFF
JSR <SCI_RCV   ; Jump to long interrupt service routine
NOP

```

```

; The SCI interrupts to here when there is an error.
ORG P:SCI_ERR,P:SCI_ERR+ROM_OFF
JSR <CLR_SCI
NOP

```

```

; DSP Timer interrupt for exposure time control
ORG P:TIM_ISR,P:TIM_ISR+ROM_OFF
JSR <TIMER     ; Long interrupt service routine
NOP

```

```

; Put the ID words for this version of the ROM code. It is placed at
; the address of the SWI = software interrupt, which we never use.
ORG P:ROM_ID,P:ROM_ID+ROM_OFF
DC $010101      ; Institution: University of Calgary
                ; Location : RAO
                ; Instrument : IR Camera
DC $000023      ; Version 0.23, board #2 = timing
                ; board #0 = video

```

```

*****
;

```

```

;
;
; Permanent address register assignments
; R1 - Address of current contents of PCI board receiver
; R2 - Address of processed contents of PCI board receiver
; R3 - Address of current contents of PCI board transmitter
; R4 - Address of processed contents of PCI board transmitter

```

```

; R6 - CCD clock driver address for CCD #0          *
; It is also the A/D address of analog board #0    *
; R7 - Return address after exposure calls, may be used sparingly *
;
; Other registers                                  *
; R0, and R5 - Temporary registers used all over the place *
;*****
;
; Initialization code is in the application area since it executes only once
;   ORG   P:APL_ADR,P:APL_ADR+ROM_OFF   ; Download address
;
; Define this as simple jump addresses so bootrom program is sure to work
; until the application program can be loaded
APPLICATION
;   JMP   <TST_RCV   ; Defined so compiler has APPLICATION address
;
; Initialization of the DSP - system register, serial link, interrupts.
; This is executed once on DSP boot from ROM, and is not incorporated
; into any download code since its not needed.
INIT  MOVEC  #$0002,OMR      ; Operating Mode Register = Normal
;                               ; Expanded - set after reset by hardware
;
;   ORI   #$03,MR           ; Temporarily mask interrupts
;
;   MOVEP  #0,X:PBC   ; Set Port B to general purpose I/O
;   MOVEP  #$3FFF,X:PBDDR      ; Set PB0 to PB14 to outputs -
;                               ; H0, AUX4, TXD_EN, RXD_EN, STATUS0 to
;                               ; STATUS3, AUX1, LVEN, AUX3, FRAME,
;                               ; LINE, AUX2. PWRST is an input.
;
;   MOVEP  #$020D,X:PBD   ; RXD-EN = TXD-EN = 1 for enabling PCI
;                               ; communication. H0 = 1 to communicate
;                               ; with analog boards. LVEN = 1.
;                               ; All others = 0.
;
;   MOVEP  #$6002,X:CRA   ; SSI programming - no prescaling;
;                               ; 24 bits/word; on-demand communications;
;                               ; no prescale; 3.12 MHz serial clock rate
;
;   MOVEP  #$3930,X:CRB   ; SSI programming - OF0, OF1 don't apply;
;                               ; SC0, SC1, SC2 are inputs; SCK is output;

```

```

; shift MSB first; rcv and xmt asynchronous
; wrt each other; gated clock; bit frame
; sync; network mode to get on-demand;
; RCV and its interrupts enabled; TX enabled,
; TX interrupts disabled -> Utility board SSI

MOVEP  #0B02,X:SCR    ; SCI programming: 10-bit asynchronous
; protocol (1 start, 8 data, no parity,
; 1 stop); LSB before MSB; enable receiver
; and its interrupts; transmitter interrupts
; disabled.

MOVEP  #0050,X:SCCR    ; SCI clock: asynchronous data rate =
; 9600 kbits/sec, internal clock.
; (50 MHz / 64 / 81 = 9645 baud)

MOVEP  #0013,X:PCC     ; Port C implemented as enabling the SCI
; pins RXD and TXD and HVEN. The SSI will
; be enabled only as needed.

MOVEP  #0013,X:PCD     ; Port C Data Register - Set all lines high
; if configured as outputs.

MOVEP  #007F,X:PCDDR   ; Port C Data Direction register - Set all
; lines to outputs when not used for SSI
; or SCI service except SRD and STD that
; are pulled low by 500 ohms.

MOVEP  #0181,X:BCR     ; Wait states = X: Y: P: and Y: ext. I/O

MOVEP  #>2,X:TCSR      ; Enable timer interrupts

MOVEP  #61A8,X:TCR     ; Divide so timer interrupts every millisecond

; Initialize X: data memory
MOVE  #RD_X,R0         ; Starting X: address in EEPROM
MOVE  #0,R1            ; Put values starting at beginning of X:
DO    #100,X MOVE      ; Assume 256 = 100 values exist
DO    #3,X LOOP        ; Reconstruct bytes to 24-bit words
MOVE  P:(R0)+,A2       ; Get one byte from EEPROM
REP   #8
ASR   A                ; Shift right 8 bits

```

```

X_LOOP
    MOVE  A1,X:(R1)+    ; Write 24-bit words to X: memory
X_MOVE

; Initialize registers
    MOVE  #RCV_BUF,R1    ; Starting address of receiver buffer
    MOVE  #XMT_BUF,R3    ; Starting address of transmitter buffer
    MOVE  #WRSS,R6       ; Address of clock and video processor switches
    MOVE  R1,R2
    CLR   A R3,R4
    MOVE  #31,M1         ; All address registers are circular, modulo 32
    MOVE  M1,M2
    MOVE  M2,M3
    MOVE  M3,M4
    MOVE  M1,N1
    DO    #32,ZERO_X     ; Zero all buffers
    MOVE  A,X:(R1)+
    MOVE  A,X:(R3)+
ZERO_X

; Disable analog board functions
    MOVEP  X:LATCH,Y:WRLATCH

; Call Load Application #1 to get video mode loaded on boot as the default
    MOVE  #'LDA',A
    MOVE  A,X:(R1)+
    MOVE  X:<ONE,A
    MOVE  A,X:(R1)+
    MOVE  X:<CAR_RET,A
    MOVE  A,X:(R1)+

; Set interrupt priority levels
    MOVEP  #$038000,X:IPR ; Write to interrupt priority register
                                ; Exposure timer = 2
                                ; SCI = 1 = PCI board link
                                ; Host, IRQA, IRQB all disabled
    ANDI   #$FC,MR        ; Unmask all interrupt levels

; Go execute the program - initialization is over
    JMP    <CHK_HDR       ; Process the commands on the stack

; Check for program space overflow
    IF     @CVS(N,*)>$1FF

```

```

        WARN  'Internal P: memory overflow!' ; Don't overflow DSP P: space
    ENDIF

; ***** End of initialization code *****

; Put some of the code in the interrupt vector area that is not used,
; from $18 to $3B (PGM_STR), then continue on at $3E (PGM_CON).
    ORG  P:PGM_STR,P:PGM_STR+ROM_OFF ; Program start

; Test serial receiver contents
START      JSET  #TIM_BIT,X:TCSR,CHK_TIM ; If timing down go elsewhere
;
; the following sucks up a few clock cycles (maybe) but aside from that
; does no harm. What it DOES do is to allow the application program
; a chance to grab the processor before the receiver gets it. So,
; leave it in for use some other time
;
    JSET  #COM_MOD,X:STATUS,APPLICATION

TST_RCV JSR    <GET_RCV ; Get a command from the receiver stack
    JEQ  <START      ; If none, test for timer and application

; Process the receiver entry - is it in the command table?
CHK_HDR  MOVE  X:(R2)+,A ; Get the command buffer entry
    MOVE  #<COM_TBL,R0 ; Get command table starting address
    DO  #NUM_COM,END_COM ; Loop over command table
    MOVE  X:(R0)+,X1 ; Get the command table entry
    CMP  X1,A X:(R0),R5 ; Does receiver = table entry?
    JNE  <NOT_COM ; No, keep looping
    ENDDO ; Yes, restore the DO loop system registers

; Wait for the complete command and then jump to it
TST_END  MOVE  X:(R1+N1),A ; Get most recent SCI word
    MOVE  X:<CAR_RET,X0
    CMP  X0,A ; Is it = "___CR" ?
    JNE  <TST_END ; No -> keep waiting

    JMP  (R5) ; Yes -> execute the command
NOT_COM MOVE  (R0)+ ; Increment the register past the table address
END_COM

; It's not in the command table - send an error message
ERROR MOVE  X:<ERR,X0 ; Send an error message 'ERR'

```

```

        JMP    <FINISH2

; Construct a simple reply for the PCI board
FINISH    MOVE    (R2)+      ; Step over Carriage Return delimiter
END_EXP   MOVE    X:<DON,X0  ; Send a 'DON' as a reply
FINISH2   MOVE    X0,X:(R3)+ ; Put on the buffer to be transmitted

; Process transmitter buffer to see if anything needs to be sent
PRC_XMT   MOVE    R4,A      ; Address of processed transmitter contents
          MOVE    R3,X0      ; Address of current transmitter contents
          CMP     X0,A       ; Are they equal?
          JEQ     <START     ; If equal, look for receiver contents
          JMP     <XMIT      ; Needed because we're inserting timer ISR

; Check contents of receiver stack to see if a new host command has come in
GET_RCV   MOVE    R2,X0      ; Get address of processed receiver contents
          MOVE    R1,A       ; Get address of current receiver contents
          CMP     X0,A
          RTS

; Jump here on RRR and M_RA commands so R2 steps over CR delimiter
XMT_DON   MOVE    (R2)+      ; Step over "___CR" delimiter in command
          RTS

; Check for program space overflow
          IF     @CVS(N,*)>$3C
          WARN   'Error: Timer ISR overwritten at P:$3C'
          ENDIF

          ORG    P:PGM_CON,P:PGM_CON+ROM_OFF ; Step over timer ISR

; Transmit the 24-bit word to the PCI board three bytes at a time
XMIT      MOVE    X:<SRXFST,R0 ; R0 = $FFF6 = SCI first byte address
          MOVE    X:(R4)+,A
          DO      #3,SCI_SPT
SCI_XMT   JCLR    #0,X:SSR,SCI_XMT ; Continue only if SCI XMT register is empty
          MOVE    A,X:(R0)-    ; Write to SCI buffer, increment byte pointer
SCI_SPT

SCI_CR    JCLR    #0,X:SSR,SCI_CR ; Continue only if SCI XMT register is empty
          MOVEP   #$0D,X:SRX   ; Transmit a Carriage Return
          JMP     <PRC_XMT

```

```

; Start up the exposure timer and wait here until it is done
EXPOSE MOVE X:<EXP_TIM,A      ; Enter exposure time into timer's
      MOVE A,X:<TGT_TIM      ; target time
      CLR  A                  ; Zero out elapsed time
      MOVE A,X:<EL_TIM
      BSET #0,X:TCSR ; Enable DSP timer
CHK_COM JSR <GET_RCV ; Check for incoming commands
      JNE <CHK_HDR ; If received, process it normally
CHK_TIM JSET #0,X:TCSR,CHK_COM ; Wait for timer to end
      JMP (R7) ; Jump to the internal jump address

; Interrupt service routine for the SCI serial link to the PCI board
SCI_RCV MOVEC SR,X:<SAVE_SR ; Save Status Register
      MOVE R0,X:<SAVE_R0 ; Save R0
      MOVE B1,X:<SAVE_B1 ; Save B1
      MOVE X1,X:<SAVE_X1 ; Save X1
      MOVE X:<SCI_R0,R0 ; Get previous value of SCI R0
      MOVE X:<SCI_B1,B1 ; Get previous value of SCI B1
      MOVE X:(R0),X1 ; Get the byte from the SCI
      OR X1,B(R0)- ; Add byte into B1, postdecrement R0
      BTST #1,R0 ; Test for the address being $FFF3 = last byte
      JCC <MID_BYT ; Not the last byte => only restore registers
END_BYT MOVE B1,X:(R1)+ ; Put the 24-bit word in the command buffer
      MOVE X:<SRXFST,R0 ; Initialize R0 most significant byte of SCI
      MOVE #0,B1 ; Zero SCI_B1 for next SCI use
MID_BYT MOVE R0,X:<SCI_R0 ; Save SCI value of SCI address pointer
      MOVE B1,X:<SCI_B1 ; Save SCI_B1 for next SCI use
      MOVEC X:<SAVE_SR,SR ; Restore Status Register
      MOVE X:<SAVE_R0,R0 ; Restore R0
      MOVE X:<SAVE_B1,B1 ; Restore B1
      MOVE X:<SAVE_X1,X1 ; Restore X1
      RTI ; Return from interrupt service

; Interrupt service routine for the DSP timer, called every millisecond
TIMER MOVEC SR,X:<SV_SR ; Save Status Register
      MOVE B1,X:<SV_B1
      MOVE Y1,X:<SV_Y1
      MOVE X:<ONE,B
      MOVE X:<EL_TIM,Y1 ; Get elapsed time
      ADD Y1,B X:<TGT_TIM,Y1 ; Get target time
      MOVE B,X:<EL_TIM ; EL_TIM = EL_TIM + 1
      CMP Y1,B
      JLT <NO_TIM ; If (EL .GE. TGT) we've timed out

```

```

        BCLR  #0,X:TCSR          ; Disable timer
NO_TIM MOVEC X:<SV_SR,SR        ; Restore Status Register
        MOVE X:<SV_B1,B1
        MOVE X:<SV_Y1,Y1
        RTI                      ; Return from TIMER interrupt

; Read DSP or EEPROM memory ('RDM' address): read memory, reply with value
RDMEM MOVE X:(R2),R0           ; Need the address in an address register
        MOVE X:(R2)+,A          ; Need address also in a 24-bit register
        JCLR #20,A,RDX          ; Test address bit for read from P: memory
        MOVE P:(R0),X0          ; Read from Program memory
        JMP <FINISH2            ; Send out a header ID with the value
RDX  JCLR #21,A,RDY            ; Test address bit for read from X: memory
        MOVE X:(R0),X0          ; Write to X data memory
        JMP <FINISH2            ; Send out a header ID with the value
RDY  JCLR #22,A,RDR            ; Test address bit for read from Y: memory
        MOVE Y:(R0),X0          ; Read from Y data memory
        JMP <FINISH2            ; Send out a header ID with the value
RDR  JCLR #23,A,ERROR          ; Test address bit for read from EEPROM memory
        MOVE X:<THREE,X0        ; Convert to word address to a byte address
        MOVE R0,Y0              ; Get 16-bit address in a data register
        MPY X0,Y0,A             ; Multiply
        ASR A                   ; Eliminate zero fill of fractional multiply
        MOVE A0,R0              ; Need to address memory
        BSET #15,R0             ; Set bit so its in EEPROM space
        DO #3,L1RDR
        MOVE P:(R0)+,A2         ; Read each ROM byte
        REP #8
        ASR A                   ; Move right into A1
L1RDR
        MOVE A1,X0              ; FINISH2 transmits X0 as its reply
        JMP <FINISH2

; Program WRMEM ('WRM' address datum): write to memory, reply 'DON'
WRMEM MOVE X:(R2),R0           ; Get the desired address
        MOVE X:(R2)+,A          ; We need a 24-bit version of the address
        MOVE X:(R2)+,X0         ; Get datum into X0 so MOVE works easily
        JCLR #20,A,WRX          ; Test address bit for write to P: memory
        MOVE X0,P:(R0)          ; Write to Program memory
        JMP <FINISH
WRX  JCLR #21,A,WRX            ; Test address bit for write to X: memory
        MOVE X0,X:(R0)          ; Write to X: memory
        JMP <FINISH

```

```

WRY  JCLR  #22,A,WRR      ; Test address bit for write to Y: memory
      MOVE  X0,Y:(R0)     ; Write to Y: memory
      JMP   <FINISH
WRR  JCLR  #23,A,ERROR     ; Test address bit for write to EEPROM
      MOVE  X:<THREE,X1    ; Convert to word address to a byte address
      MOVE  R0,Y0          ; Get 16-bit address in a data register
      MPY   X1,Y0,A        ; Multiply
      ASR   A              ; Eliminate zero fill of fractional multiply
      MOVE  A0,R0          ; Need to address memory
      BSET  #15,R0         ; Set bit so its in EEPROM space
      MOVE  X0,A1          ; Get data from command string
      DO    #3,L1WRR       ; Loop over three bytes of the word
      MOVE  A1,P:(R0)+     ; Write each EEPROM byte
      REP   #8
      ASR   A X:<C50000,Y0 ; Move right one byte, enter delay
      DO    Y0,L2WRR       ; Delay by 12 milliseconds for EEPROM write
      REP   #4             ; Assume 50 MHz DSP56002
      NOP
L2WRR
      NOP                 ; DO loop nesting restriction
L1WRR
      JMP   <FINISH

; Read EEPROM code into DSP memory starting at P:APL_ADR - as a subroutine
LDAPPL MOVE  X:(R2)+,X0    ; Number of application program
      MOVE  X:<C600,Y0
      MPY   X0,Y0,A X:<ZERO,X1
      ASR   A X:<C300,X0
      SUB   X,A #APL_ADR,R7
      MOVE  A0,R0          ; EEPROM address = # x $600 - $300
      BSET  #15,R0         ; All EEPROM accesses are with A15=1
      DO    #APL_LEN,LD_LA2 ; Load from APL_ADR to $200
      DO    #3,LD_LA1
      MOVE  P:(R0)+,A2     ; Read from EEPROM
      REP   #8
      ASR   A
LD_LA1
      MOVE  A1,P:(R7)+     ; Write to DSP P: memory
LD_LA2

; Splice the application and boot command tables together
      MOVE  #COM_TBL,R7    ; Leave most of X: memory alone
;      DO    #32,LD_LA4     ; 16 commands, 2 entries per command

```

```

DO #48,LD_LA4 ; 24 commands, 2 entries per command
DO #3,LD_LA3
MOVE P:(R0)+,A2 ; Read from EEPROM
REP #8
ASR A
LD_LA3
MOVE A1,X:(R7)+ ; Write to DSP X: memory
LD_LA4

; Transfer Y: memory, containing waveforms and readout parameters
MOVE #0,R7 ; Start at bottom of Y: memory
DO #300-32-APL_LEN,LD_LA6 ; Update y:DSP memory
DO #3,LD_LA5
MOVE P:(R0)+,A2 ; Read from EEPROM
REP #8
ASR A
LD_LA5
MOVE A1,Y:(R7)+ ; Write to DSP Y: memory
LD_LA6
JMP <FINISH ; Return and send 'DON'

; Reset = Reboot
RST RESET ; Reset peripherals
MOVE X:<CFFFF,M0 ; Insure that its linear addressing
MOVE X:<CFFFF,M1
MOVEP X:ZERO,X:IPR ; Clear Interrupt Priority Register
MOVEP X:CFFFF,X:BCR ; Many Wait States for PROM accesses
MOVEC X:<ZERO,SP ; Clear the stack pointer
MOVEC X:<C300,SR ; Clear the Condition Code Register
MOVEC #S01,OMR ; Operating Mode Register = Reboot
NOP ; Allow one cycle delay for the remapping
JMP <$0 ; Begin bootstrap from internal ROM

; Clear error condition and interrupt on SSI receiver
CLR_SCI MOVEP X:SSR,X:RCV_ERR ; Read SCI status register
MOVEP X:SRX,X:RCV_ERR ; Read register to clear error
RTI

; Check for program space overflow into application code area
IF @CVS(N,*)>=APL_ADR
WARN 'ERROR - Boot program overflows into application code area'
ENDIF

```

```
; ***** Beginning of X: definitions *****
```

```
; Status and header ID processing words
```

```
    ORG    X:0,X:LD_X
```

```
STATUS DC    0    ; Status word
```

```
; Timer related constants
```

```
SV_SR      DC    0    ; Save for timer ISR
```

```
SV_B1      DC    0    ; Save for timer ISR
```

```
SV_Y1      DC    0    ; Save for timer ISR
```

```
; Definitions for variables needed for the interrupt service routines
```

```
SAVE_SR DC    0
```

```
SAVE_X1 DC    0
```

```
SAVE_B1 DC    0
```

```
SAVE_R0 DC    0
```

```
SCI_B1 DC    0
```

```
SCI_R0 DC    $FFF6 ; Current address of the SCI
```

```
SRXFST DC    $FFF6 ; Address of first byte in SCI receiver
```

```
SPARE      DC    0
```

```
; EXP_TIM must be at address $E for compatability with IR Labs' IREM
```

```
EL_TIM      DC    0    ; Elapsed exposure time in milliseconds
```

```
TGT_TIM      DC    0    ; TGT_TIM = EXP_TIM at beginning of exposure
```

```
EXP_TIM      DC    1000 ; Exposure time (milliseconds), written by host computer
```

```
; Definition of value in latch U25
```

```
LATCH       DC    $10    ; Value in latch chip U25
```

```
; Miscellaneous constant definitions
```

```
ZERO DC    0
```

```
ONE  DC    1
```

```
TWO  DC    2
```

```
THREE DC    3
```

```
EN_SI DC    $0173    ; Enable the SCI and SSI pins
```

```
DISA_SI DC    $0003
```

```
C300 DC    $300    ; Constant for resetting the DSP
```

```
C600 DC    $600    ; EEPROM space per application program
```

```
CFFFF DC    $FFFF    ; Constant for resetting the DSP
```

```
C50000 DC    50000    ; Delay for WRROM = 12 millisec
```

```
ERR DC    'ERR'    ; An error occurred
```

```
DON DC    'DON'    ; Command was fully processed
```

```
RCV_ERR DC    0    ; Dummy location for receiver clearing
```

CAR RET DC \$20200D ; Carriage Return marking end of command

```
; Command table resident in X: data memory
; The last part of the command table is not defined for "bootrom"
; because it contains application-specific commands
```

```

    ORG      X:COM_TBL,X:COM_TBL+LD_X
DC     0,START           ; This is where application
DC     0,START           ; commands go
DC     0,START
DC     0,START
DC     0,START
DC     0,START
DC     0,START
DC     0,START
DC     0,START
DC     0,START
DC     0,START
DC     0,START
DC     0,START
DC     0,START
DC     0,START
DC     0,START
DC     0,START
DC     0,START
DC     0,START
DC     0,START
DC     0,START
DC     0,START
DC     0,START
DC     0,START
DC     0,START
DC     0,START
DC     0,START
DC     0,START
DC     0,START
DC     0,START
DC     0,START
        ; everything below this has commands
        ; intrinsic to the bootstrap
DC     'ERR',START       ; Nothing special
DC     'RDM',RDGMEM      ; Read from DSP or EEPROM memory
DC     'WRM',WRMEM       ; Write to DSP memory
DC     'LDA',LDAPPL      ; Load application program from EEPROM to DSP
DC     'RST',RST         ; Re-boot DSP from on-board ROM
DC     'STP',FINISH      ; Put it here as a no op
DC     $20200D,START     ; Extra delimiters - do nothing
DC     0,START          ; Room for one more command

```

; End of command table

```
; End of program  
END
```

D.2.4 App.asm as modified by the University of Calgary

COMMENT *

This file is used to generate DSP code for the second generation timing boards to operate a PICNIC infrared array.
 Fiber optic and PCI application files are now joined into one.
 File modified 11/97 to generate timing waveforms similar to old versions of PICNIC delivered with the microscopes
 Changed Aug. '98 to control Rev. 6C power control board, and use extended on-board SRAM

Base code University of Calgary.

Code is proved to work. Date is 010705

Required command-line switches, DL = 0; FOPCI = 0;

Modified 2001-07-9 by Anna Johnson, Physics Department,
 University of Calgary, to support a Rockwell TCM1000 infrared array.
 This is REV 0.23 software.

CHANGE HISTORY:

2001-07-09

Added the following ops:

SEX: Set exposure time.

TST: return test message to host.

LDW: direct load of word from host to WRSS.

OSH: open shutter

CSH: close shutter

Removed the following ops:

VD1: Set video mode 1

VD2: Set video mode 2

Left in but unnecessary:

STP: Stop video mode. This is left in only for the moment, and represents the only spare place in the command table.

The above ops and their associated code were cut out because while the TCM chip is intended to be run in stare mode (for this pass) it will not be run in video mode.

The exposure code was also modified so that it adds the shutter flag (SH_MASK) to the outgoing clock-generator board word to keep the shutter where it is supposed to be during the exposure. Added the following stubs:
 SV2: Set DC offset on the ADC board
 SGN: Set the gain on the ADC board.
 These will have to remain stubs until the ADC board has been wrung out.
 All UC additions are bracketed and identified.
 Designated REV 0.20, stored as OP3.

2001-07-16

Set APP_LEN to \$300 to allow for extended length programs
 Increased command table size to 32 ops.
 Designated REV 0.21, stored as BIGTAB

2001-07-18

Cut out timing tables for PICNIC array, inserted tables for TCM1000C array. Rewrote RD_ARRAY routine to accomodate new tables.
 Replaced RESET_ARRAY routine with FLUSH_ARRAY routine
 Designated REV 0.22, stored as TIMING1

2001-07-26

Added COF and CON ops to force camera power on and off.
 Rewrote solenoid throws so that they -> should <- work correctly.
 Verified that power supply is in fact a timPC and that the bit designators that are active in this code really do flip HVEN and LVEN.
 Designated REV 0.23, stored as PWRTIME

-d DL 1 To generate code for downloading to DSP memory.
 -d DL 0 To generate code for writing to the EEPROM.

*

PAGE 132 ; Printronix page width - 132 columns

; Define a section name so it doesn't conflict with other application programs
 SECTION TIMIR

```
; These are the equates which determine which and how many analog channels get
; transmitted.
```

```
SXMIT0_3 EQU $00F060      ; Series transmit A/D channels #0 - 3
SXMIT0 EQU $00F000        ; Transmit ADC 0 only
```

```
SXMIT EQU SXMIT0
```

```
; These are the assignments of the lines on the clock generator board.
```

```
; LINE 0 CLK $0 PIN 1
; LINE 1 YSYN $1 PIN 2
; LINE 2 FRAME $2 PIN 3
; LINE 4 POWER $4 PIN 4
; LINE 3 Close Shutter $8 PIN 5
; LINE 4 Close Shutter hold $10 PIN 6
; LINE 5 Open Shutter $20 PIN 7
; LINE 6 Open Shutter Hold $40 PIN 8
```

```
; Shutter bit designators
```

```
; these get used in BSET and BCLR statements
```

```
CLK_BIT EQU $0
YSYN_BIT EQU $1
FRAME_BIT EQU $2
POWER_BIT EQU $3
CLOSE_BIT EQU $4
CLOSE_HOLD_BIT EQU $5
OPEN_BIT EQU $6
OPEN_HOLD_BIT EQU $7
```

```
; Shutter bit patterns.
```

```
; These get used in the wave tables
```

```
; Chip timing low and high are defined just before the wave tables.
```

```
UC_POWER EQU $4 ; Power word
UC_SCLS EQU $8 ; UC close shutter word
UC_SCHD EQU $10 ; UC close shutter hold word
UC_SOPN EQU $20 ; UC open shutter word
UC_SOHD EQU $40 ; UC open shutter hold word
```

```
; Clock board voltage definitions. These definitions assume +VREF = 5 volts,
```

```
; -VREF = 0.
```

```
CLK_HI EQU $FFF
CLK_LO EQU $000
SHUT_HI EQU $FFF ; full load
```

```

SHUT_LO    EQU    $07A    ; holding voltage = 0.3 volts
SHUT_OFF   EQU    $000

;
; board assignments for clock generator boards
BD1        EQU    $001000
BD2        EQU    $002000

; assign the active clock generator board
TB         EQU    BD2

; video board stuff
OFF_MASK0A EQU    $0c0000 ; offset mask, video board 0, channel A
GAIN_MASK0A EQU    $003000 ; gain mask, video board 0, channel A

; Miscellaneous definitions
VIDEO EQU    $000000      ; Video board select = 0 for first A/D board with
                           ; biases
DELAY EQU    $480000      ; Delay for clocking operations
                           ; (20 ns unit, 160 if MSB set)
VP_DLY EQU    $2C0000      ; Video delay time for 3 microsec/pixel

; These are also defined in "timboot.asm", so be sure they agree
APL_NUM     EQU    1      ; Application number from 1 to 10
APL_ADR     EQU    $F0    ; start application code at beginning of assigned block
APL_LEN     EQU    $300-APL_ADR ; Maximum length of application program
COM_TBL     EQU    $A0    ; Starting address of command table in X: memory
TBL_ADR     EQU    $0F    ; Waveform tables starting address
NUM_COM     EQU    32     ; total number of commands in comm table
                           ; of which 24 belong to the application.
                           ; 8 belong to the loader

; Define some timing board addresses and bit numbers
WRFO        EQU    $FFC0  ; Write to fiber optic serial transmitter
WRLATCH     EQU    $FFC1  ; Write to timing board latch
SSITX       EQU    $FFEF  ; SSI Transmit and Receive data register
PCC         EQU    $FFE1  ; Port C Control Register
PBD         EQU    $FFE4  ; Port B Data Register
TCSR        EQU    $FFDE  ; Timer control and status register
CDAC        EQU    0      ; Bit number in U25 for clearing DACs
WW          EQU    1      ; Word width of serial data

```

```

ENCK      EQU 2      ; Bit number in U25 for enabling analog switches
;LVEN     EQU 2      ; Low voltage enable (+/- 6.5, +/- 15 volt nominal)
;HVEN     EQU 3      ; High voltage enable (+36 volts, only used for reset)

```

```

; Values for timPC board

```

```

LVEN      EQU 9      ; Low voltage enable (+/- 6.5, +/- 15 volt nominal)
HVEN      EQU 4      ; High voltage enable (+36 volts, only used for reset)

```

```

; Specify execution and load addresses

```

```

IF DL
  ORG P:APL_ADR,P:APL_ADR      ; Download address
ELSE
  ORG P:APL_ADR,P:(2*APL_NUM-1)*$100 ; EEPROM generation
ENDIF

```

```

APPLICATION

```

```

  JMP <TST_RCV

```

```

; Exit video mode, enter continuous reset mode

```

```

STP BSET #RST_MOD,X:STATUS      ; Continuous reset mode on
  BCLR #VID1_MOD,X:<STATUS
  BCLR #VID2_MOD,X:<STATUS
  JMP <FINISH

```

```

; University of Calgary additions to code start here

```

```

; Set the exposure timer value

```

```

UC_SEX MOVE X:(R2)+,A          ; get datum into X0 so MOVE works easily
  MOVE #>5,X0                  ; Subtract 5 millisec from exposure time to      ; clipped
AJ 010709
  SUB X0,A                      ; account for READ to FSYNC delay time      ; clipped
AJ 010709
  MOVE A,X:<EXP_TIM              ; load the desired exposure time
  JMP <FINISH                    ; and exit

```

```

;

```

```

; Send a test message

```

```

UC_TST MOVE Y:<TST,X0          ; load reply
  JMP <FINISH2                  ; and send

```

```

;

```

```

; Load test word to WRSS port (parallel output on backplane)

```

```

UC_LDW
  MOVE X:(R2)+,X0              ; get datum into X0 so move works easily
  MOVE X0,X:WRSS                ; direct write to port
  JMP <FINISH

```

```

;
; open the shutter
UC_OSH JSR  UCOSHS          ; this is made a subroutine so it
      JMP  <FINISH          ; can be used in the exp routine
;
; open shutter routine
UCOSHS
      BCLR  #CLOSE_BIT,Y:<SH_MASK      ; clear the CLOSE bit
      BCLR  #CLOSE_HOLD_BIT,Y:<SH_MASK ; clear the hold close bit
      BSET  #OPEN_BIT,Y:<SH_MASK       ; set the open bit
      MOVE  Y:<SH_MASK,B1              ; load the mask
      MOVE  Y:<YTB,Y1
      OR    Y1,B1                     ; insert the board address
      MOVE  Y:<YDELAY,Y1
      OR    Y1,B1                     ; insert the delay
      MOVE  B1,X:WRSS                 ; write to port
      JSR  <SHORT_DELAY               ; wait for shutter to swing
      BCLR  #OPEN_BIT,Y:<SH_MASK       ; clear the open bit
      BSET  #OPEN_HOLD_BIT,Y:<SH_MASK  ; set the hold open bit
ORBLOCK                               ; lots of routines use this
      MOVE  Y:<SH_MASK,B1              ; logic, need to trim some fat
      MOVE  Y:<YTB,Y1
      OR    Y1,B1                     ; insert the board address
      MOVE  Y:<YDELAY,Y1
      OR    Y1,B1                     ; insert the delay
      MOVE  B1,X:WRSS                 ; write to port
      RTS                             ; and exit
;
; close the shutter
UC_CSH JSR  UCCSHS          ; this is made a subroutine so it can
      JMP  <FINISH          ; be used in the exposure routine
;
; close shutter subroutine
UCCSHS
      BCLR  #OPEN_BIT,Y:<SH_MASK       ; clear the OPEN bit
      BCLR  #OPEN_HOLD_BIT,Y:<SH_MASK  ; clear the hold open bit
      BSET  #CLOSE_BIT,Y:<SH_MASK      ; set the close bit
      MOVE  Y:<SH_MASK,B1              ; load the mask
      MOVE  Y:<YTB,Y1
      OR    Y1,B1                     ; insert the board address
      MOVE  Y:<YDELAY,Y1
      OR    Y1,B1                     ; insert the delay
      MOVE  B1,X:WRSS                 ; write to port

```

```

JSR <SHORT_DELAY          ; wait for shutter to swing
BCLR #CLOSE_BIT,Y:<SH_MASK ; clear the close bit
BSET #CLOSE_HOLD_BIT,Y:<SH_MASK ; set the hold close bit
JMP <ORBLOCK
; MOVE Y:<SH_MASK,B1
; MOVE Y:<YTB,Y1
; OR Y1,B1          ; insert the board address
; MOVE Y:<YDELAY,Y1
; OR Y1,B1          ; insert the delay
; MOVE B1,X:WRSS    ; write to port
; RTS              ; and exit
;
; Do a simple read of the array for diagnostic purposes
; will result in DMA transfer
UC_RDAR JSR <SHORT_DELAY    ; need at least 1 millisecond delay
JSR <RD_ARRAY              ; read the array
JMP <FINISH                ; send kissoff to PCI card
;
; change bias voltage on ADC card
UC_SB2
MOVE X:(R2)+,X1            ; get datum into X0 so move works easily
MOVE Y:<OFF_0,B1            ; get mask from storage
MOVE Y:<OFFSET,Y1           ; get bare mask
AND Y1,B1                  ; clear out old stuff
OR X1,B1                   ; move in new stuff
MOVE B1,Y:<OFF_0            ; move to offset value storage
JSR <SETBIAS               ; set the bias as requested
JMP <FINISH                ; and exit
;
; change gain on ADC card
UC_SGN
MOVE X:(R2)+,X1            ; get datum into X0 so move works easily
MOVE Y:<GAIN_0,B1           ; get mask from storage
MOVE Y:<GAIN,Y1             ; get bare mask
AND Y1,B1                  ; clear out old stuff
OR X1,B1                   ; move in new stuff
MOVE B1,Y:<GAIN_0           ; move to offset value storage
JSR <SETBIAS               ; set the bias as requested
JMP <FINISH                ; and exit
;
; End University of Calgary additions
;

```

```

;
; Short delay to allow array to settle down after reset.
SHORT_DELAY
    MOVE Y:<RST_DLY,A          ; Enter reset delay into timer
CON_DELAY          ; Alternate entry for camera on delay
    MOVE A,X:<TGT_TIM
    CLR A              ; Zero out elapsed time
    MOVE A,X:<EL_TIM
    BSET #0,X:TCSR      ; Enable DSP timer
CNT_DWN
    JSET #0,X:TCSR,CNT_DWN    ; Wait here for timer to count down
    RTS

; Abort exposure and stop the timer
ABR_EXP
    CLR A              ; Just stop the timer
    MOVE A,X:<TGT_TIM
    JMP <FINISH         ; Send normal reply

; Dummy subroutine to not call receiver checking routine
NO_CHK
    BCLR #0,SR          ; Clear status register clear bit
    RTS

; ***** ARRAY READOUT *****
RD_ARRAY
    IF FOPCI            ; Optionally send "RDA" to FO-PCI board
        MOVEP Y:CBD_HDR,Y:WRFO
        JSR <PAL_DLY
        MOVEP Y:RDA,Y:WRFO
        JSR <PAL_DLY
        MOVEP Y:NPXLS,Y:WRFO
    ENDIF

; below inserted 010718 AJ.
; this routine is set up for the TCM1000B with single-channel output,
; all 128 lines with FRAME line high.
;
    BSET #WW,X:PBD       ; Set WW to 1 for 16-bit image data
    DO #598,DLY_ON       ; count levels
    JSR <PAL_DLY
    NOP
DLY_ON

```

```

DO #128, GROUP_1 ; read first set with frame line asserted.
MOVE #<READ_ON, R0 ; generate initial pulse
JSR <CLOCK

DO #128, GROUP_1_INNER ; clock out the pixels
MOVE #<FIRST_64, R0 ; get the drive sequence
JSR <CLOCK ; clock it out
NOP ; loop restriction
GROUP_1_INNER

MOVE #<READ_OFF, R0 ; last pulse
JSR <CLOCK
NOP ; loop restriction
GROUP_1

JSR <PAL_DLY ; Wait for serial data transmission
BCLR #WW, X:PBD ; Clear WW to 0 for non-image data
RTS
; endit

```

```

; ***** ARRAY FLUSH *****
;
FLUSH_ARRAY
IF FOPCI ; Optionally send "RDA" to FO-PCI board
MOVEP Y: CBD_HDR, Y: WRFO
JSR <PAL_DLY
MOVEP Y: RDA, Y: WRFO
JSR <PAL_DLY
MOVEP Y: NPXLS, Y: WRFO
ENDIF

```

below inserted 010718 AJ.

this routine is set up for the TCM1000B with single-channel output,
all lines with FRAME line high.

FLUSH_ARRAY has everything RD_ARRAY has, except that it does not transmit.

```

BSET #WW, X:PBD ; Set WW to 1 for 16-bit image data
DO #598, FDLY_ON ; count levels
JSR <PAL_DLY
NOP
FDLY_ON

```



```

L_MRA1
    MOVE X:<ZERO,A1      ; clear A1 register
    MOVE X:<EXP_TIM,A0    ; get exposure time
    TST  A                ; test for zero
    JEQ  L_MRANO          ; don't open the shutter if zero
    JSR  <UCOSHS          ; Open the shutter
    JSR  <SHORT_DELAY     ; wait for it to swing over
L_MRANO
    MOVE #L_MRA2,R7      ; load return address
    JMP  <EXPOSE          ; Delay for specified exposure time
L_MRA2
    JSR  <UCCSHS          ; Close the shutter
    JSR  <SHORT_DELAY     ; wait for it to swing over
    DO   Y:<N_RA,L_MRA3    ; Read N_RA times again
    JSR  <RD_ARRAY        ; Call read array subroutine
    NOP                      ; will result in DMA transfer
L_MRA3
    JMP  <END_EXP         ; This is the end of the exposure

; ***** SUBROUTINES *****
; Core subroutine for clocking out array charge
CLOCK
    MOVE Y:SH_MASK,Y1    ; get the shutter hold mask
    MOVE Y:(R0)+,X0      ; # of waveform entries
    MOVE Y:(R0)+,A        ; start the pipeline
    DO   X0,CLK1          ; repeat X0 times
    OR   Y1,A            ; insert shutter hold mask
    MOVE A,X:(R6) Y:(R0)+,A ; send out the waveform
CLK1
    OR   Y1,A            ; insert shutter hold mask
    MOVE A,X:(R6)        ; flush out the pipeline
    RTS                      ; and return
;
; Update the DACs
SET_DAC
    DO   Y:(R0)+,SET_L0   ; Repeat X0 times
    MOVEP Y:(R0)+,X:SSITX ; Send out the waveform
    JSR  <PAL_DLY         ; Wait for SSI and PAL to be empty
    NOP                      ; Do loop restriction
SET_L0
    RTS                      ; Return from subroutine

; Delay for serial writes to the PALs and DACs by 8 microsec

```

```

PAL_DLY
    DO #200,DLY          ; Wait 8 usec for serial data transmission
    NOP
DLY
    NOP
    RTS

; Delay between power control board instructions
DLY_PWR
    DO #4000,L_PDLY
    NOP
L_PDLY
    RTS

; Set video offsets and DC bias supply voltages
; This may be dead code for the moment, but it is left in so
; that if we have more than one ADC active, the code can be used.
; Leave this clip in place. 010719 AJ
; SET_BIAS_NUM
;     MOVEP X:EN_SI,X:PCC          ; Enable the SSI
;     JSR <PAL_DLY                ; Wait for the SSI port to be enabled
;     MOVE X:(R2)+,A              ; First argument is board number, 0 to 15
;     REP #20
;     LSL A
;     MOVE A,X0
;     MOVE X:(R2)+,A              ; Second argument is DAC number, 0 to 7
;     REP #14
;     LSL A
;     OR X0,A
;     BSET #19,A1                 ; Set bits meaning DAC
;     BSET #18,A1
;     MOVE A,X0
;     MOVE X:(R2)+,A              ; Third argument is voltage value
;     MOVE #000FFF,Y0            ; Mask off just 12 bits to be sure
;     AND Y0,A
;     OR X0,A
;     MOVEP A,X:SSITX             ; Write the number to the DAC
;     MOVEP X:DISA_SI,X:PCC       ; Disable the SSI
;     JMP <FINISH
; endit

; Power off
PWR_OFF

```

```

    MOVEP X:EN_SI,X:PCC      ; Enable the SSI
    BCLR #CDAC,X:<LATCH      ; Clear all DACs
    BCLR #ENCK,X:<LATCH      ; Disable DAC output switches
    MOVEP X:LATCH,Y:WRLATCH
    BSET #LVEN,X:PBD         ; LVEN = HVEN = 1 => Power reset
;   BSET #HVEN,X:PBD         ; !!!
    BSET #HVEN,X:PCD         ; timPC value
    BCLR #COM_MOD,X:STATUS   ; Command execution mode
    MOVEP X:DISA_SI,X:PCC    ; Disable the SSI
    BCLR #POWER_BIT,Y:<SH_MASK ; clear bit in shutter mask
    JSR <ORBLOCK              ; try to trim some fat
;   MOVE Y:<SH_MASK,B1        ; get the mask
;   MOVE Y:<YTB,Y1
;   OR Y1,B1                  ; put in the clock gen board addr
;   MOVE Y:<YDELAY,Y1
;   OR Y1,B1                  ; put in the delay
;   MOVE B1,X:WRSS            ; direct write to port
    JMP <FINISH

; Start power-on cycle
CAM_ON
    MOVEP X:EN_SI,X:PCC      ; Enable the SSI
    BSET #CDAC,X:<LATCH      ; Disable clearing of all DACs
    BCLR #ENCK,X:<LATCH      ; Disable DAC output switches
    MOVEP X:LATCH,Y:WRLATCH

; Turn analog power on to controller boards, but not yet to IR array
    BSET #LVEN,X:PBD         ; LVEN = HVEN = 1 => Power reset
;   BSET #HVEN,X:PBD         ; !!!
    BSET #HVEN,X:PCD         ; timPC value

; Now ramp up the low voltages (+/- 6.5V, 16.5V) and delay them to turn on
    BCLR #LVEN,X:PBD         ; LVEN = Low => Turn on +/- 6.5V, +/- 16.5V
    MOVE Y:<PWR_DLY,A
    JSR <CON_DELAY

; applying power to the IR array is split out as a separate operation.

; Set DC bias DACs
SETBIAS
    MOVEP X:EN_SI,X:PCC      ; Enable the SSI
    JSR <PAL_DLY              ; Wait for port to be enabled
    MOVE #<DC_BIASES,R0      ; Get starting address of DAC values

```

```

    JSR <SET_DAC
    MOVE X:<THREE,A          ; Delay three millisec to settle
    JSR <CON_DELAY

; Set clock driver DACs
    MOVE #<TB_DACS,R0        ; Get starting address of DAC values
    JSR <SET_DAC

; disable the SSI and return
    MOVEP X:DISA_SL,X:PCC    ; Disable the SSI
    JMP <FINISH

UC_CON                                ; Now, apply power to the IR array
    BSET #POWER_BIT,Y:SH_MASK ; sets the shutter mask
    JSR <ORBLOCK              ; trim some fat
;   MOVE Y:<SH_MASK,B1        ; get updated mask
;   MOVE Y:<YTB,Y1            ; get board address
;   OR Y1,B1
;   MOVE Y:<YDELAY,Y1         ; get delay
;   OR Y1,B1
;   MOVE B1,X:WRSS            ; and write to port

; Delay for the IR array to settle down
    MOVE Y:<VDD_DLY,A         ; Delay for the IR array to settle
    JSR <CON_DELAY
    JMP <FINISH

UC_COF
    BCLR #POWER_BIT,Y:SH_MASK ; clears the bit in the shutter mask
    JSR <ORBLOCK              ; trim some fat
;   MOVE Y:<SH_MASK,B1        ; get updated mask
;   MOVE Y:<YTB,Y1            ; get board address
;   OR Y1,B1
;   MOVE Y:<YDELAY,Y1         ; get delay
;   OR Y1,B1
;   MOVE B1,X:WRSS            ; and write to port
    JMP <FINISH              ; and get the hell out of Dodge.

; check to make sure that code does not overflow into data.
    IF !DL

```

```

IF @CVS(N,@LCV(L))>(2*APL_NUM-1)*$100+APL_LEN
WARN 'CODE overflow!' ; Make sure next application
ENDIF ; will not be overwritten
ENDIF

; Command table
IF DL ; Memory offsets for downloading code
ORG X:COM_TBL,X:COM_TBL
ELSE ; Memory offsets for generating EEPROMs
ORG P:COM_TBL,P:(2*APL_NUM-1)*$100+APL_LEN
ENDIF

; Begin UC command table
DC 0,START ; dummy 0
DC 0,START ; dummy 1
DC 0,START ; dummy 2
DC 0,START ; dummy 3
DC 0,START ; dummy 4
DC 0,START ; dummy 5
DC 'COF',UC_COF ; turn off power to the IR array
DC 'CON',UC_CON ; turn on power to the IR array
DC 'PON',CAM_ON ; turn on all clocks
DC 'POF',PWR_OFF ; turn +/- 15V power supplies off.
DC 'SBV',SETBIAS ; Set DC bias supply voltages
DC 'SB2',UC_SB2 ; set ADC board bias voltage
DC 'SGN',UC_SGN ; set ADC board gain selector
DC 'STP',STP ; exit VIDEO mode
DC 'RDA',UC_RDAR ; Read out array once for diagnostics
DC 'RRR',RRR ; Reset, read, read array
DC 'MRA',M_RA ; multiple reads of array
DC 'ABR',END_EXP ; end current exposure
DC 'DON',START ; nothing special
DC 'TST',UC_TST ; reply with test message
DC 'SEX',UC_SEX ; set exposure time
DC 'LDW',UC_LDW ; Load word to parallel port out (diagnostic)
DC 'OSH',UC_OSH ; open shutter
DC 'CSH',UC_CSH ; close shutter
; endit

IF DL
ORG Y:0,Y:0 ; Download address
ELSE
ORG Y:0,P: ; EEPROM address continues from P: above

```

```

ENDIF
;
;UC additions 2001-07-07
TST DC 'TST' ; test message reply
SH_MASK DC 0 ; shutter mask

DUMMY DC 0 ; Left over from previous versions
NCOLS DC 63 ; Number of columns
NROWS DC 64 ; Number of rows
N_RA DC 1 ; Desired number of reset/read pairs
RST_DLY DC 50 ; Delay after array reset for settling
PWR_DLY DC 100 ; Delay in millisec for power to turn on
VDD_DLY DC 300 ; Delay in millisec for VDD to settle
N_RSTS DC 2 ; Number of resets
NPXLS DC 65536 ; Number of pixels transmitted per image
CBD_HDR DC $AA0002 ; Header to transmit to converter board
RDA DC 'RDA' ; Read array command

; the below is a bit touchy, but the objective is to have OFF_0 at Y:$10
YTB DC TB
YDELAY DC DELAY
OFFSET DC OFF_MASK0A
GAIN DC GAIN_MASK0A

; Table of offset values begins at Y:$10
; well, maybe not.

; DAC settings for the video offsets
DC_BIASES
DC END_BIASES-DC_BIASES-1
OFF_0 DC $0c0000 ; Input offset board #0, channel A
GAIN_0 DC $003000 ; input gain, board #0, channel A
END_BIASES
DC 0 ; place holder

TB_DACS
DC TB_DACS_END-TB_DACS-1
DC (TB<<8)+(0<<14)+CLK_HI ; Pin #1, CLK
DC (TB<<8)+(1<<14)+CLK_LO
DC (TB<<8)+(2<<14)+CLK_HI ; Pin #2, YSYN
DC (TB<<8)+(3<<14)+CLK_LO

```

```

DC    (TB<<8)+(4<<14)+CLK_HI  ; Pin #3, FRAME
DC    (TB<<8)+(5<<14)+CLK_LO
DC    (TB<<8)+(6<<14)+CLK_HI  ; Pin #4, SPARE
DC    (TB<<8)+(7<<14)+CLK_LO
DC    (TB<<8)+(8<<14)+SHUT_HI  ; Pin #5, CLOSE
DC    (TB<<8)+(9<<14)+SHUT_OFF
DC    (TB<<8)+(10<<14)+SHUT_LO ; Pin #6, CLOSE HOLD
DC    (TB<<8)+(11<<14)+SHUT_OFF
DC    (TB<<8)+(12<<14)+SHUT_HI ; Pin #7, OPEN
DC    (TB<<8)+(13<<14)+SHUT_OFF
DC    (TB<<8)+(14<<14)+SHUT_LO ; Pin #8, OPEN HOLD
DC    (TB<<8)+(15<<14)+SHUT_OFF
TB_DACS_END
DC    0                      ; end DAC table

```

```

; UC table for TCM1000B. This table assumes 128 x 128 readout, single
; channel readout, frame line asserted for first 64 lines then
; deasserted for last 64 reads.

```

```

; Define switch state bits for the clocks

```

```

CLK_L EQU $0    ; defines clock as line 0 on clock generator board
CLK_H EQU $1
YSYN_L EQU $0    ; defines YSYN as line 1 on clock generator board
YSYN_H EQU $2
FRAME_L EQU $0   ; defines FRAME as line 2 on clock generator board
FRAME_H EQU $4
POWER_L EQU $0   ; defines POWER as line 3 on clock generator board
POWER_H EQU $8
CLOSE_L EQU $0   ; defines CLOSE as line 4 on clock generator board
CLOSE_H EQU $10
CH_L EQU $0      ; defines CLOSE HOLD as line 5 on clock generator board
CH_H EQU $20
OP_L EQU $0      ; defines OPEN as line 6 on clock generator board
OP_H EQU $40
OPH_L EQU $0     ; defines OPEN HOLD as line 7 on clock generator board
OPH_H EQU $80

```

```

READ_ON      ; Initial pulse

```

```

DC    FIRST_64-READ_ON-2
DC    BD2+DELAY+CLK_H+YSYN_H+FRAME_H
DC    BD2+DELAY+CLK_L+YSYN_H+FRAME_H

```

```

;
FIRST_64      ; first 64 lines with frame line asserted

```

```

DC SECOND_64-FIRST_64-2
DC BD2+DELAY+CLK_H+YSYN_L+FRAME_H ; toggle clock up
DC BD2+DELAY+CLK_L+YSYN_L+FRAME_H ; toggle clock down
DC VP_DLY ; A/D sample
DC $010033 ; Start A/D conversion
DC SXMIT ; Series transmit four pixels' data
DC $140033 ; Delay
;
SECOND_64 ; second 64 lines with frame line deasserted
DC READ_OFF-SECOND_64-2
DC BD2+DELAY+CLK_H+YSYN_L+FRAME_L ; toggle clock up
DC BD2+DELAY+CLK_L+YSYN_L+FRAME_L ; toggle clock down
DC VP_DLY ; A/D sample
DC $010033 ; Start A/D conversion
DC SXMIT ; Series transmit four pixels' data
DC $140033 ; Delay
;
READ_OFF ; trailing pulse to make up 130 pixels (per line) read
DC FLUSH_FIRST-READ_OFF-2
DC BD2+DELAY+CLK_H+YSYN_L+FRAME_L ; final clock high
DC BD2+DELAY+CLK_L+YSYN_L+FRAME_L ; final clock low
;
FLUSH_FIRST ; timing sequence to flush first 64 without transmit
DC FLUSH_SECOND-FLUSH_FIRST-2
DC BD2+DELAY+CLK_H+YSYN_L+FRAME_H ; toggle clock up
DC BD2+DELAY+CLK_L+YSYN_L+FRAME_H ; toggle clock down
;
FLUSH_SECOND ; timing sequence to flush second 64 without transmit
DC END_TAB-FLUSH_SECOND-2
DC BD2+DELAY+CLK_H+YSYN_L+FRAME_L ; toggle clock up
DC BD2+DELAY+CLK_L+YSYN_L+FRAME_L ; toggle clock down
;
END_TAB
DC 0 ; end UC table
; endit
;
; Check for overflow in the EEPROM case
IF !DL
IF @CVS(N,@LCV(L))>((2*APL_NUM+1)*$100)*3
WARN 'EEPROM overflow!' ; Make sure next application
ENDIF ; will not be overwritten
ENDIF

```

ENDSEC ; End of section TIMIR

; End of program
END

D.2.5 Assembly script for Motorola Assembler Under MS-DOS

```
asm56000 -v -b -lboot.ls boot.asm
asm56000 -v -b -d DL 0 -d FOPCI 0 -lapp.ls app.asm
dsplnk -btcm.cld -v boot.cln app.cln
del tcm.lod
del tcm.p
del tcm.x
cldlod tcm.cld > tcm.lod
srec -b -s tcm.lod
```

APPENDIX E: Host Program Error Codes

ABEND OCCURRED, ERROR CODE IS <code>

This is a fatal error. It should never happen but if it does happen it is because something happened in the message cracker. Please report the incident and the message code to the software engineer.

CANNOT AUTOLOAD POSITION - FUNCTION NOT IMPLEMENTED

This is a non-fatal error. The host program cannot get position information from the telescope positioning computer.

CANNOT FIND TEST PATTERN

This is a non-fatal error. The test patterns are not in the current working directory.

CONFIG READ: FILE ERROR

This is a non-fatal error. The configuration file could not be read. Ensure that the current working directory has a copy of the configuration file.

CONFIG READ: GET LOCAL DIRECTORY FAILED

This is a non-fatal error. The specified local directory could not be found.

Ensure that the directory does exist and try again.

CONFIG READ: I/O ERROR ON GET LOCAL DRIVE

This is a non-fatal error. The specified local drive could not be accessed. Ensure that you have specified a valid drive and try again.

DMA TRANSFER ERROR

This is a non-fatal error. The DMA transfer from the IR Labs controller failed.

Try again.

ERROR ON CREATING SUBDIRECTORIES

This is a non-fatal error. You tried to create a file system for image storage and the attempt failed. The usual reason is because a subdirectory of the same name already exists.

ERROR - FILESET HAS NOT BEEN INITIALIZED

This is a non-fatal error. You tried to save images without first creating or opening a fileset.

FATAL ERROR - BAD SELECT IN MAIN SWITCH

This is a fatal error. It should never happen but if it does something happened in the CVIRT engine that caused it to return a bad value to the host program message cracker. Please report the incident to the software engineer.

FEATURE NOT IMPLEMENTED THIS VERSION

This is a non-fatal error. You tried to use a feature of the host program that was allocated space when the program was set up, but has not yet been implemented.

FITS FILE CREATION ERROR

This is a non-fatal error. The FITS file handler cannot create a file.

FITS IMAGE CREATION ERROR

This is a non-fatal error. The FITS file handler cannot save your image. This is usually caused by bad data in the location or identification fields.

FITS WRITE IMAGE ERROR

This is a fatal error. The FITS file handler cannot write your image to disk.

Please report the incident to the software engineer.

NO HELP FORKING

This is a non-fatal error. The help file could not be found.

NO HIST FORKING

This is a non-fatal error. The history file could not be found.

NO LEGAL FORKING

This is a non-fatal error. The legal file could not be found.

READ_CONFIG: INVALID FILE NAME

This is a non-fatal error. You specified an invalid name for the configuration file.

SPECTRAL SYSTEMS BOARD ERROR

This is a non-fatal error. Something is amiss with the DMA interface to the IR Labs controller. Record the error code and report the incident to the software engineer.

WRITE_CONFIG: INVALID FILE NAME

This is a non-fatal error. You specified an invalid name for the configuration file.

APPENDIX F: CFITSIO ERROR STATUS CODES

The tables below are taken from Reference 9. Permission to freely use, copy, modify, and distribute the software and its documentation is granted in the documentation, provided that the following copyright notice and disclaimer of warranty appear in all copies.

DISCLAIMER:

“THE SOFTWARE IS PROVIDED ‘AS IS’ WITHOUT ANY WARRANTY OF ANY KIND, EITHER EXPRESSED, IMPLIED, OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, ANY WARRANTY THAT THE SOFTWARE WILL CONFORM TO SPECIFICATIONS, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND FREEDOM FROM INFRINGEMENT, AND ANY WARRANTY THAT THE SOFTWARE WILL BE ERROR FREE. IN NO EVENT SHALL NASA BE LIABLE FOR ANY DAMAGES, INCLUDING, BUT NOT LIMITED TO, DIRECT, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES, ARISING OUT OF, RESULTING FROM, OR IN ANY WAY CONNECTED WITH THIS SOFTWARE, WHETHER OR NOT BASED UPON WARRANTY, CONTRACT, TORT, OR OTHERWISE, WHETHER OR NOT INJURY WAS SUSTAINED BY PERSONS OR PROPERTY OR OTHERWISE, AND WHETHER OR NOT LOSS WAS SUSTAINED FROM, OR AROSE OUT OF THE RESULTS OF, OR USE OF, THE SOFTWARE OR SERVICES PROVIDED HEREUNDER.”

CFITSIO Error Status Codes

The following table lists all the error status codes used by CFITSIO. Programmers are encouraged to use the symbolic mnemonics (defined in the file fitsio.h) rather than the actual integer status values to improve the readability of their code.

Symbolic Const	Value	Meaning
	0	OK, no error
PREPEND_PRIMARY	-9	used in ffiting to prepend a new primary array
SAME_FILE	101	input and output files are the same
TOO_MANY_FILES	103	tried to open too many FITS files at once
FILE_NOT_OPENED	104	could not open the named file
FILE_NOT_CREATED	105	could not create the named file
WRITE_ERROR	106	error writing to FITS file
END_OF_FILE	107	tried to move past end of file
READ_ERROR	108	error reading from FITS file
FILE_NOT_CLOSED	110	could not close the file
ARRAY_TOO_BIG	111	array dimensions exceed internal limit
READONLY_FILE	112	Cannot write to readonly file
MEMORY_ALLOCATION	113	Could not allocate memory
BAD_FILEPTR	114	invalid fitsfile pointer
NULL_INPUT_PTR	115	NULL input pointer to routine
SEEK_ERROR	116	error seeking position in file
BAD_URL_PREFIX	121	invalid URL prefix on file name
TOO_MANY_DRIVERS	122	tried to register too many IO drivers
DRIVER_INIT_FAILED	123	driver initialization failed
NO_MATCHING_DRIVER	124	matching driver is not registered
URL_PARSE_ERROR	125	failed to parse input file URL
SHARED_BADARG	151	bad argument in shared memory driver
SHARED_NULLPTR	152	null pointer passed as an argument
SHARED_TABFULL	153	no more free shared memory handles

SHARED_NOTINIT	154	shared memory driver is not initialized
SHARED_IPCERR	155	IPC error returned by a system call
SHARED_NOMEM	156	no memory in shared memory driver
SHARED_AGAIN	157	resource deadlock would occur
SHARED_NOFILE	158	attempt to open/create lock file failed
SHARED_NORESIZE	159	shared memory block cannot be resized at the moment
HEADER_NOT_EMPTY	201	header already contains keywords
KEY_NO_EXIST	202	keyword not found in header
KEY_OUT_BOUNDS	203	keyword record number is out of bounds
VALUE_UNDEFINED	204	keyword value field is blank
NO_QUOTE	205	string is missing the closing quote
BAD_KEYCHAR	207	illegal character in keyword name or card
BAD_ORDER	208	required keywords out of order
NOT_POS_INT	209	keyword value is not a positive integer
NO_END	210	couldn't find END keyword
BAD_BITPIX	211	illegal BITPIX keyword value
BAD_NAXIS	212	illegal NAXIS keyword value
BAD_NAXES	213	illegal NAXISn keyword value
BAD_PCOUNT	214	illegal PCOUNT keyword value
BAD_GCOUNT	215	illegal GCOUNT keyword value
BAD_TFIELDS	216	illegal TFIELDS keyword value
NEG_WIDTH	217	negative table row size
NEG_ROWS	218	negative number of rows in table
COL_NOT_FOUND	219	column with this name not found in table
BAD_SIMPLE	220	illegal value of SIMPLE keyword
NO_SIMPLE	221	Primary array doesn't start with SIMPLE
NO_BITPIX	222	Second keyword not BITPIX
NO_NAXIS	223	Third keyword not NAXIS
NO_NAXES	224	Couldn't find all the NAXISn keywords
NO_XTENSION	225	HDU doesn't start with XTENSION keyword
NOT_ATABLE	226	the CHDU is not an ASCII table extension
NOT_BTABLE	227	the CHDU is not a binary table extension
NO_PCOUNT	228	couldn't find PCOUNT keyword
NO_GCOUNT	229	couldn't find GCOUNT keyword
NO_TFIELDS	230	couldn't find TFIELDS keyword
NO_TBCOL	231	couldn't find TBCOLn keyword
NO_TFORM	232	couldn't find TFORMn keyword
NOT_IMAGE	233	the CHDU is not an IMAGE extension
BAD_TBCOL	234	TBCOLn keyword value < 0 or > rowlength
NOT_TABLE	235	the CHDU is not a table
COL_TOO_WIDE	236	column is too wide to fit in table
COL_NOT_UNIQUE	237	more than 1 column name matches template
BAD_ROW_WIDTH	241	sum of column widths not = NAXIS1
UNKNOWN_EXT	251	unrecognizable FITS extension type
UNKNOWN_REC	252	unknown record; 1st keyword not SIMPLE or XTENSION

END_JUNK	253	END keyword is not blank
BAD_HEADER_FILL	254	Header fill area contains non-blank chars
BAD_DATA_FILL	255	Illegal data fill bytes (not zero or blank)
BAD_TFORM	261	illegal TFORM format code
BAD_TFORM_DTYPE	262	unrecognizable TFORM datatype code
BAD_TDIM	263	illegal TDIMn keyword value
BAD_HDU_NUM	301	HDU number < 1 or > MAXHDU
BAD_COL_NUM	302	column number < 1 or > tfields
NEG_FILE_POS	304	tried to move to negative byte location in file
NEG_BYTES	306	tried to read or write negative number of bytes
BAD_ROW_NUM	307	illegal starting row number in table
BAD_ELEM_NUM	308	illegal starting element number in vector
NOT_ASCII_COL	309	this is not an ASCII string column
NOT_LOGICAL_COL	310	this is not a logical datatype column
BAD_atable_FORMAT	311	ASCII table column has wrong format
BAD_btable_FORMAT	312	Binary table column has wrong format
NO_NULL	314	null value has not been defined
NOT_VARI_LEN	317	this is not a variable length column
BAD_DIMEN	320	illegal number of dimensions in array
BAD_PIX_NUM	321	first pixel number greater than last pixel
ZERO_SCALE	322	illegal BSCALE or TSCALn keyword = 0
NEG_AXIS	323	illegal axis length < 1
NOT_GROUP_TABLE	340	Grouping function error
HDU_ALREADY_MEMBER	341	
MEMBER_NOT_FOUND	342	
GROUP_NOT_FOUND	343	
BAD_GROUP_ID	344	
TOO_MANY_HDUS_TRACKED	345	
HDU_ALREADY_TRACKED	346	
BAD_OPTION	347	
IDENTICAL_POINTERS	348	
BAD_GROUP_ATTACH	349	
BAD_GROUP_DETACH	350	
NGP_NO_MEMORY	360	malloc failed
NGP_READ_ERR	361	read error from file
NGP_NUL_PTR	362	null pointer passed as an argument. Passing null pointer as a name of template file raises this error
NGP_EMPTY_CURLINE	363	line read seems to be empty (used internally)
NGP_UNREAD_QUEUE_FULL	364	cannot unread more than 1 line (or single line twice)
NGP_INC_NESTING	366	too deep include file nesting (infinite

		loop, template includes itself ?)
NGP_ERR_FOPEN	366	fopen() failed, cannot open template file
NGP_EOF	367	end of file encountered and not expected
NGP_BAD_ARG	368	bad arguments passed. Usually means internal parser error. Should not happen
NGP_TOKEN_NOT_EXPECT	369	token not expected here
BAD_I2C	401	bad int to formatted string conversion
BAD_F2C	402	bad float to formatted string conversion
BAD_INTKEY	403	can't interpret keyword value as integer
BAD_LOGICALKEY	404	can't interpret keyword value as logical
BAD_FLOATKEY	405	can't interpret keyword value as float
BAD_DOUBLEKEY	406	can't interpret keyword value as double
BAD_C2I	407	bad formatted string to int conversion
BAD_C2F	408	bad formatted string to float conversion
BAD_C2D	409	bad formatted string to double conversion
BAD_DATATYPE	410	illegal datatype code value
BAD_DECIM	411	bad number of decimal places specified
NUM_OVERFLOW	412	overflow during datatype conversion
DATA_COMPRESSION_ERR	413	error compressing image
DATA_DECOMPRESSION_ERR	414	error uncompressing image
BAD_DATE	420	error in date or time conversion
PARSE_SYNTAX_ERR	431	syntax error in parser expression
PARSE_BAD_TYPE	432	expression did not evaluate to desired type
PARSE_LRG_VECTOR	433	vector result too large to return in array
PARSE_NO_OUTPUT	434	data parser failed not sent an out column
PARSE_BAD_COL	435	bad data encounter while parsing column
PARSE_BAD_OUTPUT	436	Output file not of proper type
ANGLE_TOO_BIG	501	celestial angle too large for projection
BAD_WCS_VAL	502	bad celestial coordinate or pixel value
WCS_ERROR	503	error in celestial coordinate calculation
BAD_WCS_PROJ	504	unsupported type of celestial projection
NO_WCS_KEY	505	celestial coordinate keywords not found
APPROX_WCS_KEY	506	approximate wcs keyword values were returned

Postlude:**Ultima facta est. Gloria in excelsis Deo.**